

COMPUTER AIDED DESIGN OF TWO LAYER PCB

**A Thesis Submitted
In Partial Fulfilment of the Requirements
for the Degree of
MASTER OF TECHNOLOGY**

by

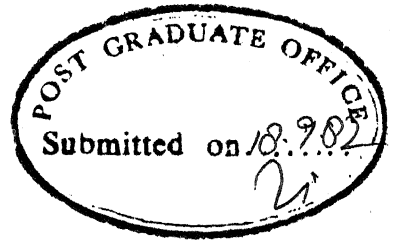
FLT. LT. L. N. RAO

to the

COMPUTER SCIENCE PROGRAMME

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

SEPTEMBER, 1982



CERTIFICATE

Certified that the thesis entitled 'COMPUTER AIDED DESIGN OF TWO LAYER PCB' by Flt.Lt. L.N. RAO has been carried out under my supervision and this has not been submitted elsewhere for a degree.

15 Sept., 1982

H.V. Sahasrabudhe
(H.V. SAHASRABUDDHE)
Professor
Department of Computer Science
Indian Institute of Technology
KANPUR

28 MAY 1984

28 MAY 1984

CENTRAL LIBRARY

Acc. No. **A.....82583**

C.S.P-1982-M-RAO-COM

ACKNOWLEDGEMENTS

It is a pleasure indeed to put my pen down on to paper to write this piece because of the inevitable feeling that a service personnel, who entered rigid academics after a long gap, should have on the realization that this is the culmination of that academic trudging of two years.

This is not to suggest that the period of this institution have converted me to a truant. Far from it. In the student - teacher relationship that we had to have these two years, these gentlemen have certainly inspired me to persevere in the process of learning.

So, it is with thanks that I recall the 'Gurus' of mine: Dr. HV Sahasrabuddhe, Dr. R. Sankar, Dr. KV Nori, Dr. MR Sridharan Dr. B. Sarkar, Dr. SG Dhande, Dr. SN Biswas, Dr. A.S. Sethi, Mr. Nirmal Roberts and the younger and brighter classmates of mine.

I have particularly enjoyed working on this thesis thanks to the ever present suggestions and encouragement from Dr. Sahasrabuddhe and Dr. KC Gupta. I hope this humble implementation would be shaped better.

Thanks are certainly due to Mr. J.S. Rawat for his efficient typing and lastly (with the traditional 'but not in the least'!) to my wife Sheila who bore the brunt of this course for my sake.

IIT Kanpur
Sep. 1982

Flt.Lt. L.N. Rao

ABSTRACT

This thesis deals with computer aided solution to the problem of 'placement' of components, and the 'Routing' of wires of interconnection between components with regard to a two layer printed circuit board. The main concern of the thesis is the implementation (in Pascal) of a force-vector heuristic technique for automated placement of components on a PCB and integrate it with an existing router with slight modifications, as also a concise review of both the placement and routing problems with suggestions for future implementations.

The problem is of importance since miniaturization, in addition to making the wire routing difficult also makes wirelength a major contributor to signal degradation when coupled with the fact of high switching speeds in circuitry. Therefore both the placement problem and routing problem have minimization/reduction of total weighted wire length as their objective functions. The routing and placement problems are always dealt with separately because of computational complexity of the total problem.

The first chapter introduces the problem and terminology. Next two chapters review the placement and routing problem in general. The placement problem using heuristic technique, its implementation and integration with the router is covered in further three chapters. The concluding chapter contains the suggestions for future implementation.

CONTENTS

CHAPTER		Page
1	INTRODUCTION AND TERMINOLOGY	1
	1.1 Terminology	2
	1.2 Chapterwise organization	3
2	THE PLACEMENT PROBLEM OVERVIEW	5
	2.1 Problem Definition	5
	2.2 Approaches to Solution	5
	2.3 Constructive Methods	6
	2.3.1 Selection rules	7
	2.3.2 Positioning rules	9
	2.4 Iterative Methods	10
	2.4.1 Abstract Algorithm	11
	2.5 Branch-and-Bound Methods	12
3	ROUTING PROBLEM OVERVIEW	16
	3.1 Wire List Determination	17
	3.2 Layering	17
	3.3 Ordering	17
	3.4 Wire Layout: Lee's Algorithm	18
	3.4.1 Backtracking	19
	3.5 Some Speedup Techniques	20
	3.6 Storage Problems	22
4	A FORCE VECTOR HEURISTIC TECHNIQUE FOR PLACEMENT	23
	4.1 Modelling Considerations	23
	4.1.1 Module types	24
	4.1.2 Example of Board Modelling	26
	4.2 Net Interconnection Rules	27
	4.3 Program overview	27

4.4	Converge Process	29
4.4.1	Vector forces on a Module and Calculation of Resultant	30
4.4.2	Repulsion vectors	32
4.4.3	Module Types and Vectors	34
4.4.4	Movement of modules	34
4.4.5	Net Interconnection rules and its Frequency of implementation	36
4.4.6	Progressive addition of nets and initial conditioning	37
4.4.7	Stopping Condition	38
4.5	Expand Process	39
4.5.1	Reducing physical overlap	40
4.5.2	Other considerations	41
4.5.3	Stopping condition	42
4.6	Final Assignment	42
4.6.1	The nature of solution	43
4.6.2	Hungarian Method	43
4.6.3	Steps of the Hungarian Algorithm	44
5	PLACEMENT IMPLEMENTATION	47
5.1	Data Input	48
5.2	Data Output	52
5.3	Program Overview	56
5.3.1	Steps of the process	56
5.3.2	Some important variables and constants	60
5.3.3	Brief notes on procedures	65
5.4	Use of Mic File for Program Execution	70

CHAPTER	Page
6 ROUTER IMPLEMENTATION	72
6.1 Data Input	72
6.2 Data Output	75
6.3 Plotting Routine	77
6.4 Program Overview	78
6.4.1 Steps of the process	78
6.4.2 Some important variables and constants	84
6.4.3 Brief Notes on Procedures	86
6.5 Mic File Instructions for Program Execution	88
7 CONCLUSION	90
7.1 Placement	90
7.2 Routing	91
7.3 Suggestion for Future Implementation	92
BIBLIOGRAPHY AND REFERENCES	94
APPENDICES	

LIST OF APPENDICES

APPENDIX NUMBER	SUBJECT
I	LISTING OF INPUT FOR PLACEMENT (CIRCUIT)
II	LISTING OF PLACEMENT PROGRAM
III	LISTING OF PLACEMENT OUTPUT (CIRCUIT)
IV	LISTING OF ROUTER PROGRAM
V	LISTING OF ROUTER OUTPUT (SAME AS FOR21-DAT)
VI	LISTING OF PLOTTING ROUTINE
VII	LISTING OF FOUR MIC FILE INSTRUCTIONS FOR EXECUTION OF ROUTINES FOR CIRCUIT1 (PLACE1, ROUTE1, PLOT1, AND NAME1).
VIII	PLACEMENT PLOT OF CIRCUIT1
IX	ROUTING PLOT OF CIRCUIT1
X	PLACEMENT PLOT OF CIRCUIT2
XI	ROUTING PLOT OF CIRCUIT2

CHAPTER 1

INTRODUCTION AND TERMINOLOGY

This thesis deals mainly with Automated placement of multiterminal electronic components in a Printed Circuit Board Environment. The problem is of importance due to two reasons: Firstly the pace of development in integrated circuit electronics is rapidly reducing the physical sizes of components, at the same time increasing the switching speeds available. This makes the wirelength a major contributor to signal degradation and the layout must minimize the wirelength. Second reason is also due to miniaturisation, which makes the wire routing difficult. The solution is sought at the placement stage itself, which (as an aid to routing) must also satisfy the connectability requirements at the same time that it tries to minimize the wirelength.

A two layer PCB consists basically of a 'Board', i.e. the area allowed for placement of electronic components. Each component, whatever be its internal mechanism, provides some external pins. Depending upon the circuit function different subsets of these pins are to be inter connected. The wires of such connection contribute to the total wirelength of the PCB. The wirelengths may be weighted.

The production of the PCB involves making an artmaster which in turn is to be made from a scale drawing of the components and the electrical inter connections. This project is an attempt to make such a drawing with the aid of computer, given the description of the board, the components and the inter connections with the objective of minimizing the total wirelength.

Whether such a drawing is attempted with the help of the computer or manually the basic problems confronting the designer are the same, namely 'PLACEMENT' AND 'ROUTING'. The two are inter related but treated separately because of the inherent computational complexity of the total problem.

1.1 TERMINOLOGY

'Board': A board is the physical area for placement of components of the circuit and for accommodating the wires inter connecting them. It is defined by its span in X and Y directions or by a set of corner points.

'Modules' : A module refers to any element that needs to be assigned to the board with a specific location, i.e. a specific X and Y coordinates. A module may have several pins that are electrically isolated but bound physically together by the module. All pins common to a single module

are treated as the same pin. As a component, the term module may refer to any of these: Package, Chip, IC or Connector.

'Nets' or 'Signalsets': These describe the inter connections of modules. A Net consists of all electrically common pins on a board. Each net may employ its own wiring or inter connection rule specified.

1.2 CHAPTERWISE ORGANIZATION

The following five chapters deal with the subject matter as follows:

Chapter 2: An overview of the placement problem. The placement problem is defined and three approaches to solution are discussed in essence.

Chapter 3: An overview of the routing problem. Brief discussion about which of the pins to be connected in what order and how. The emphasis is on Lee's algorithm for actual routing between pin pairs.

Chapter 4: Discussion in detail of a force-vector heuristic technique of placement. The implementation (in Pascal) of this technique is the main concern of this thesis.

Chapter 5: The pascal implementation of the placement program is discussed from user's point of view and documented.

Chapter 6: An existing routing program [10] is documented for ease of user.

Chapter 7: Conclusion.

CHAPTER 2

THE PLACEMENT PROBLEM OVERVIEW

2.1 PROBLEM DEFINITION

Formally, the module placement problem consists of finding the optimal placement of modules in the board with respect to some norm defined on the interconnections.

Since interconnecting wirelengths is a major contributor to signal degradation, especially in an environment of miniature components, it must be reduced. There are also a number of other circuit goals: minimum wirebuildup in the routing channels, elimination of signal echoes, minimum heat dissipation etc. Since for practical purposes it is impossible to directly incorporate all of the circuit goals, the total weighted wirelength is used as an approximation in trying to satisfy them enmasse.

Apart from reduction of wirelength the placement should also have as its another primary aim as that of aiding the wire routing. Therefore in general we can state that the placement problem is that of assignment of M modules to N sites that satisfies the connectability requirements and also minimizes the total interconnecting distances.

2.2 APPROACHES TO SOLUTION

Because of the complexity and magnitude of most

practical problems no guaranteed method for optimum solution exists. Breuer's formulation of the problem as an integer linear programming task [2] provides an exact solution but the constraints imposed make the solution of less practical value. Therefore methods based on heuristic rationales are employed. They may be classified as follows.

'Constructive Techniques': Useful for initial placement. A placement configuration is produced only upon termination of algorithm.

'Iterative Techniques': Useful for placement improvement. At every stage there is a complete placement available. The algorithm improves it by repeated modification.

'Branch and Bound Techniques': These are also constructive techniques in essence.

2.3 CONSTRUCTIVE METHODS

In this placement configuration is formed by adjoining modules to a subset of already placed modules. The input data consists of details of board, nets, modules and other control parameters and iteratively selects one of the unplaced modules for inclusion in the set of placed modules. Once placed the modules are not moved.

These methods have the advantage of requiring relatively small amount of computer time and yet sufficient

for many applications. The specific methods under this category are defined by the particular rules of 'selection' (i.e. ordering of modules) and 'positioning' of modules. The known methods are: 'Pair-Linking Methods' and 'Cluster Development Methods' due to J.M. Kurtzberg.

2.3.1 SELECTION RULES

To determine the order of module selection some measure of connectivity of the modules is required i.e., some notion of how 'strongly' unplaced modules are bound to already placed modules. A natural measure of connectivity between two distinct modules is sum of weights of all nets in which both these distinct modules are members. Let us call that measure for two distinct modules X and Y as $P(X,Y)$.

One selection 'function' may be based as follows. It will consider each module X included in the set of unplaced modules and each module Y included in the set of placed modules and select that module X for placement which has $\max P(X,Y)$. A second rule may incorporate the connectivity from an unplaced module X to all placed modules Y and select such X which gives $\max \sum P(X,Y)$.

Yet another selection rule may be based on the number of nets to which unplaced module X belongs and select such X which has maximum of its nets in the process of

placement. A net is considered in the process of placement if atleast one of its modules but not all are already placed. A fourth rule may improve upon it and select such X whose nets together contribute the maximum number of already placed modules. Yet another rule may improve it by taking the number of modules in each net of X into consideration in the same rule.

In general a selection rule does not yield a unique module. In that case a 'tie-breaking' rule is necessary which can be: (a) Look-ahead small number of steps;
(b) Optimise on a secondary function or
(c) Choose arbitrarily.

The above rules define the generic types but many other meaningful rules can be defined. The particular application, interdependence of net and computer time etc. dictate the choice of a selection rule.

The selection rules described are all independent of the positioning rule. Hence the modules can be completely ordered, if desired, before the positioning rule is applied. But when the selection rule depends upon the relative positioning of the modules on board the computation becomes more complex.

Note that when a module has all the modules of all its nets unplaced, that module is not a candidate for the next iteration of selection rule.

2.3.2 POSITIONING RULES

In positioning a module, we try to find the best slot for it relative to the modules already positioned. Instead of examining all empty slots, only a comparatively small subset of slots ('candidate slots') are considered. These subsets may be accumulated along with the positioning of each new module or an appropriate set may be regenerated for each positioning. For example, a simple pattern of four adjacent slots of each newly placed module can be added. As each slot is occupied it is deleted from the candidate set and the slots already occupied are never re-employed and unusable slots are never added to the set. In this pattern the periphery of positioned modules is always the candidate set. H. Freitag and M. Hanan of IBM have used a variation of this technique.

With each candidate slot accommodating the module to be positioned, we compute the wirelength (cost) of all the (partial) nets of which this module is a member, based on that particular nets interconnection rule. The module can then be placed in that candidate slot which gives the minimum cost.

In practice this can not always be done due to two reasons: one is excessive computer time and the other is that since not all modules in each of these nets have

been placed, the optimal interconnection can not be determined at this point. Thus an approximation rule is frequently used. For example, if the rule is minimum spanning tree then for each candidate slot we may compute the distance to the closest module in each of the partially placed nets considered and we choose that slot which gives the minimum total distance.

2.4 ITERATIVE METHODS

This project concerns the implementation of a heuristic technique of applying vector forces to component modules and iteratively relocating them to arrive at a hopefully optimal placement. This iterative method, implemented in Pascal, is based on a IEE.ACM Design Automation Workshop paper [2]. This method which is independent of initial placement will be discussed in detail in Chapter 4. Other iterative methods which have been tried mainly as an improvement of initial placement are:

- (a) Steinberg's Assignment Method
(Steinberg and Rutnam 1964)
- (b) Relaxation Method (Casey and West 1967)
- (c) Pairwise interchange Method (Kurtzberg and Hanan 1970)
- (d) Stochastic Method or 'Montecarlo' Approaches
(Cooper 1964, Kurtzberg 1960).

2.4.1 ABSTRACT ALGORITHM

This class of algorithms in general, abstract sense can be formulated in the following paragraphs.

Step 1: Generate initial placement configuration.

Step 2: Generate a 'transformation' whereby the layout is mapped to a new configuration.

Step 3: Compute the cost of this new configuration and compare it with the old configuration.

Step 4: If the cost (may be wirelength) of the new layout is reduced the new configuration is substituted for the old one. Otherwise the old placement is retained.

Step 5: Apply 'stopping rule' with outcomes:

(a) Go to step 2. i.e. continue modifying the placement.

The basis to terminate this loop depends upon the rate of improvement in the cost or upon the predetermined number of transformations without improvement.

(b) Go to step 1. i.e. a new starting point is made with a new initial placement for modification. The decision to stop this loop depends on the time available.

(c) Stop.

The resultant placement for each initial placement is 'locally optimum' in the sense that there does not exist a placement with a smaller cost for the given initial configuration.

We can generate a large number of locally optimum placements, thereby increasing the probability that atleast one of these placements is close to the global optimum.

Alternatively we can find better locally optimum placements thereby increasing the probability that each initial placement leads to a placement close to the optimum.

2.5 BRANCH-AND-BOUND METHODS

These methods have the feature that they can be used to find an optimum solution (when the placement problem is viewed as a quadratic assignment problem) if the order is less than 15 modules. Gilmore (1962) and Lawler (1963) have independently applied these methods with Gilmore modelling an approximation to the exact branch and bound method.

Briefly stated, the branch-and-bound method proceeds as follows. The set of all feasible solutions is partitioned and a search for the optimum is made in each partition. A lower bound is computed for the solutions in each partition, and the search in any partition is terminated upon the lower bound exceeding the cost of some previously found feasible solution.

The feasible solution may be obtained by any method, e.g., by a constructive method or by branch-and-bound algorithm itself.

To define methods for computing lower bounds and partitioning sets of lower bounds consider the quadratic assignment problem in which there is a connection matrix $C = [c_{ij}]$ and a distance matrix $D = [d_{ij}]$ and we seek a permutation p such that

$$\sum_{i,j} c_{ij} d_{p(i)p(j)}$$

is a minimum.

As a preliminary consider the branch and bound method from the view point of an $(n-1)$ stage decision process. At each stage k , a module i is chosen from the remaining $(n-k)$ modules and assigned to all the remaining $(n-k)$ slots. The term branch and bound comes from viewing the method as a search in a decision tree.

On the first level tree in Fig.2.1, module i has been chosen and each vertex at first level is a possible slot for this module. On the second level module i_2 is chosen and assigned to any of the remaining slots.

For each vertex in the decision tree, a lower bound is computed. Gilmore developed two lower bounds for

$\sum c_{ij} d_{p(i)p(j)}$ based on the observation that if

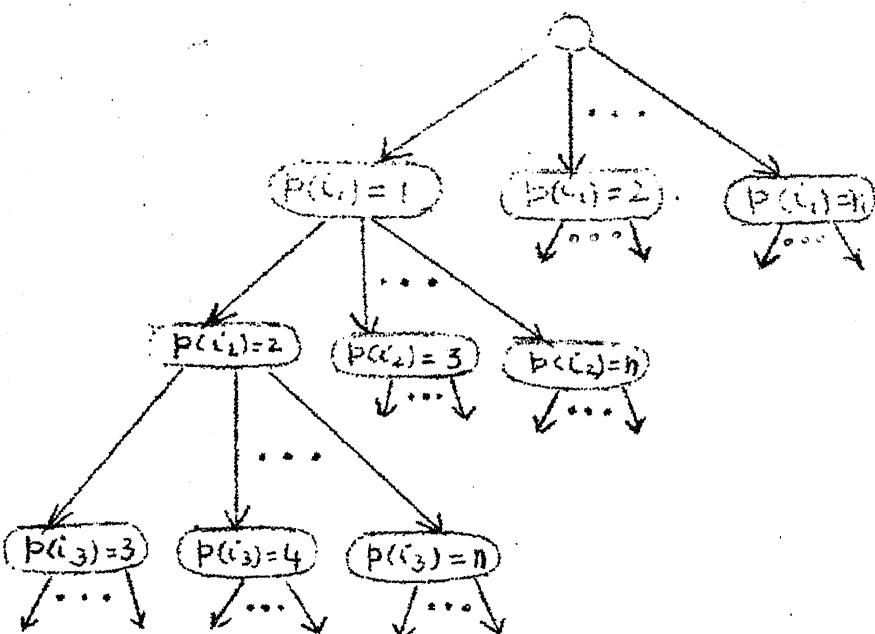


Fig. 2.1

$\bar{c} = (c_1, c_2, \dots, c_m)$ and $\bar{d} = (d_1, d_2, \dots, d_m)$ are two vectors then the minimum dot product of \bar{c} and \bar{d} i.e. $\sum c_i d_{p(i)}$ over all permutations p is obtained by arranging \bar{c} in increasing order and \bar{d} in decreasing order, and then multiplying term by term.

A simple lower bound for the quadratic assignment problem is obtained by treating the nondiagonal elements of matrices c and d as vectors of length $n(n-1)$ and performing the proper ordering and multiplication. Diagonal elements are eliminated since no pin is connected to itself and two modules can not occupy the same slot. Also if some

modules are already placed, then the contribution of these placed modules to the total distance is computed, the corresponding entries from c and d matrices are deleted and a new lower bound is calculated by forming vectors from the matrices without the entries used in the computation of placed modules contribution.

In another calculation of lower bound the problem reduces to finding the solution for assignment problem of finding the minimum of $\sum a_{ip(i)}$ over all permutations p where $[a_{ij}]$ is the 'assignment matrix' A . The element a_{ij} of matrix A is the minimum dot product of vectors \bar{c}_i and \bar{d}_i which are formed from matrices c and d after deleting entry c_{ii} and d_{jj} respectively. Since the total cost of placing module i into slot j is $\sum c_{ik} d_{jp(k)}$, a_{ij} is the lower bound on the cost of placing module i in slot j . A permutation π for which $a_{i\pi(i)}$ is minimum then yields a lower bound for the original quadratic assignment problem.

The efficiency of the branch and bound method depends on both the bounding and branching techniques. The function of the lower bound is to prune the decision tree; the better the bound the sharper the pruning. However a tradeoff is required since better the bound, the more the computation time required to compute this bound.

CHAPTER 3

ROUTING PROBLEM OVERVIEW

The process of specifying the precise conductor paths necessary to properly interconnect the module pins of each net subject to the imposed constraints of wire thickness, spacing, constraints, feedthrough reduction etc. is the crux of the problem. Our placement techniques are aids to this. We may say that the constraints stated above contribute to the weight of wirelength which we seek to reduce. Again a compromise between running time and optimality of solution must be effected. Various authors agree that typically, a 'poor' solution could be obtained in about 5 seconds of computer time and the 'best' solution in probably over 5 hours. The basic approach to the routing problem is stated in the following paragraph.

'Lee's algorithm' is the basic tool in the final layout process. But a successful layout results from a series of closely interlocked steps, each designed towards the ultimate goal of making the final layout procedure as simple and as efficient as possible. The four basic steps are:

- (a) Wire list determination;
- (b) Layering;
- (c) Ordering;
- (d) Wire layout.

Roughly speaking these are the 'what', 'where', 'when' and 'how' of the problem.

3.1 WIRE LIST DETERMINATION

This lists precisely what wires are to be connected, i.e. which pin to which pin. This is simplified if such a list is produced in the output of placement algorithm itself or it can be incorporated during the routing process. The desired wiring rule for each net, when implemented keeping the objective function in view produces a list of such pins or pin pairs for each net to be connected.

3.2 LAYERING

This involves tentatively assigning each wire to one of the layers of the given multilayer board. This is easily and arbitrarily solved in the case of two layer boards. One approach is to try to route as many wires as possible on the first side under consideration. Another approach is discussed in [8].

3.3 ORDERING

This is to determine in what order should the wires on each layer be processed. We may consider each wire to be a net in the sense we defined a net. Then it becomes a question of in what order each net is going to be laid out. There are many rules of thumb developed for this and there is

nothing sacred about a particular rule and most of them tend to lead to a nesting of wires. The significant point is that of finding paths which will cause the least difficulties for subsequent path layouts. There is no firm guarantee or precise answer to this. However, a reasonable choice seems to be that path which uses up the fewest grid segments.

3.4 WIRE LAYOUT: LEE'S ALGORITHM

The previous three topics have been specifically oriented towards how the individual net wires are to be laid out interconnecting each net. Lee's algorithm is a fundamental tool which greatly simplifies the process.

This is actually an application of the shortest path algorithm used in O.R. and graph theory. Here the paths under consideration must follow the cells of a rectangular grid and hence the algorithm becomes even simpler to apply.

The process begins with the selection of either pin as a starting point. Say A is selected (with reference to Fig. 3.1). Initially a 1 is entered in each empty cell immediately adjacent to the cell containing pin A. Next 2s are entered in all empty cells adjacent to those cells containing 1's. Next 3's are entered adjacent to 2s and so on.

The process continues in this manner until one of two results occur: Either all paths become blocked, i.e. on the

4	3	2	3	4	5	6	7	8	9	10	11
3	2	1	2	3	4		8	9	10	11	12
2	1	(A)	1	2	5		9	10	11		
		1	2		6	7	8	9	10	11	12
4	3	2	3		7					12	13
5	4				8	9	10			(B)	
6	5	6		10	9	10	11	12	13		
		7	8	9	10		12				
10	9	8	9	10	11		13				
11	10	9	10	11	12						

Fig. 3.1: Lee's Algorithm

kth step there are no empty cells adjacent to those containing (k-1) or the target cell B is reached. In this case B is reached on the 13th step indicating that there is a path from A to B and the shortest such path is of length 13. All that remains is to actually label this path.

3.4.1 BACK TRACKING

Since B was reached on the 13th step, it follows that there must be a 12 immediately adjacent to it and which in turn must be adjacent to 11 and so on. Proceeding in this manner we label the paths leading back to A. Now when we have a choice in back tracking, i.e. when there are two or more cells with (k-1) adjacent to a cell containing k,

in theory any of these cells may be chosen and path still found. In practice (reduce bends) we adopt the guideline: Don't change directions unless you have to. Also when a turn must be made we select and try the same sequence of directions for all turns to try to achieve a uniform nesting. More sophisticated nesting techniques have been tried but the program gets more and more complex.

3.5 SOME SPEEDUP TECHNIQUES

In a typical wire layout, 70 to 80 percent of running time is spent on Lee's algorithm and any technique that can be employed to speed up its execution has observable pay offs. Three commonly used techniques are:

Starting point selections: Consider the pins X and Y in Fig. 3.2. If X is chosen as starting point (Fig. 3.2(a)), less than half the total cells will have to be labelled before Y is reached. On the other hand choosing Y as starting point leads to labelling practically all the cells.

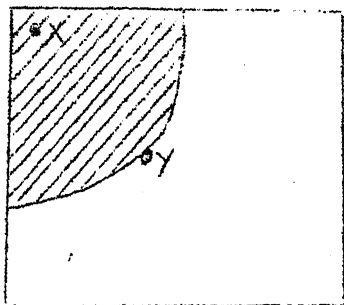


Fig. 3.2(a)

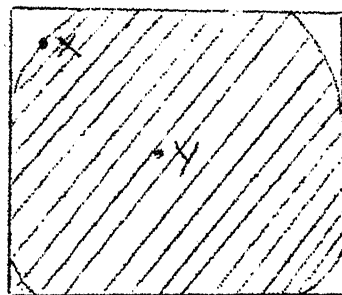


Fig. 3.2(b)

Fig. 3.2: Starting Point Selection

So the guide line can be; start from the pin farthest from the centre of the board. But this loses importance as the number of wires laid out increases.

Double fan out: In this we begin at both pins. The

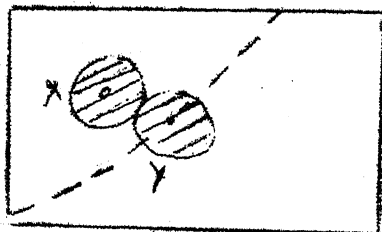


Fig. 3.3: Double Fan out

program will be much more complicated.

Framing: An artificial rectangular boundary is imposed about the pin pair being processed and no labelling

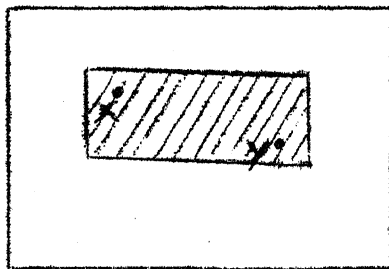


Fig. 3,4: Framing

is allowed beyond this boundary, say about 10 to 20 percent larger than the rectangle defined by the pins. When path is not found in the frame it is repeated with larger frame. If the repetition occurs too frequently towards the end the framing is discarded,

3.6 STORAGE PROBLEMS

The primary disadvantage of Lee's algorithm is that for every cell in the grid there must be a corresponding memory location. In certain modified algorithm it has been tried to avoid by having a 'list of current latest cells' and in each new step the list is updated with only those cells adjacent to cells on this list. Another economy is by economising the label each cell has to receive. When path length is long the label number is more, and more lists are required. This can be avoided by using a binary coding of only 2 bits of 3 bits for each label by utilising the fact that during the back track process, it is only necessary that 'predecessor' labels be distinguished from 'successor' labels.

CHAPTER 4

A FORCE VECTOR HEURISTIC TECHNIQUE FOR PLACEMENT

This placement technique has been implemented in pascal. A listing of which is attached as Appendix II. This is a heuristic technique of applying vector forces to modules and iteratively relocating them. The methods used to compute these forces improve the ability to find solutions that are independent of initial conditions and near optimal. This employs a very flexible modelling scheme.

The aims of this technique are:

- (a) Flexibility in handling many sizes and shapes of components.
- (b) Adherence to any wiring (interconnection) rules specified.
- (c) Minimization of total wirelength.
- (d) Improving connectability.
- (e) Handling practical problems without requiring impractical memory.
- (f) Fast and computationally efficient method.

4.1 MODELLING CONSIDERATIONS

The ideas of Board, Module and Net are retained in the

same sense as already defined. But there are six classifications of modules and three interconnection rules implemented.

4.1.1 MODULE TYPES

Free modules: Those modules that are free to assume any X and Y coordinate necessary for their placement. Only the boundaries of the board and interference of other modules restrict their positioning. The majority of the modules are usually of this free type.

Fixed modules: These modules can not be moved. Their X and Y coordinates are specified in the input data and remain so. The terminal connectors for example are of this type.

Fixed-X modules: These are modules whose X-coordinate must remain constant but may assume any specific Y coordinate during the execution of the program.

Fixed-Y modules: Similar to fixed-X. But here the Y coordinate must remain fixed. The program may alter its X-coordinate. Normally some stand-alone discrete component or other special circuit requirement may give occasion for a fixed X or fixed Y module.

X-bar modules: The X-bar modules are like the fixed-X modules only as far as the fact that their X coordinate must

remain constant. But their Y coordinate instead of being a single point appears as a vertical line of indefinite length to the program. Any connection to such a module is merely a perpendicular to that line.

Y-bar modules: These are the counter parts of X-bar modules with their Y coordinate constant and appearing as a horizontal line of indefinite length to the program. Bar modules generally apply to a set of identical components occupying a certain fixed area and the connections to which from any point of the board is likely to be a perpendicular line to that general area.

Every module has a physical size. Sizes are rectangular dimensions of length and width from the centre point i.e. coordinate position of the module. For the placement program modules are modelled as single dimensionless points (i.e. coordinate position) and pins of a module are also treated as the same point.

The terminal connector plug can be 'segmented' to avoid too many modules that would arise by treating each board terminal pin as a module. 'Segmenting' is simply grouping pins of the connector plug into areas that are approximately the size of the other modules on the board.

To model irregular shape of boards extra sites may be defined by adding fixed type dummy modules. The dummy modules have no connections and no assignment would be made to them, but will exert a physical force on the positioning.

4.1.2 EXAMPLE OF BOARD MODELLING

In this board the origin of coordinate system can be assumed at left most bottom corner. Each integer

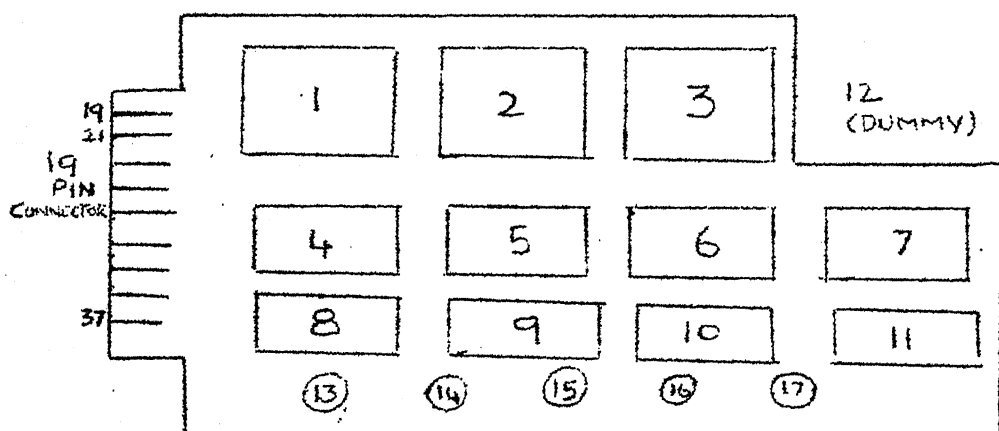


Fig. 4.1: Circuit board to be modelled

coordinate unit along the axes may correspond to $\frac{1}{10}$ " or 1 or 2 mm as desired. The coordinate positions (centres) and sizes of all modules are in integers. Modules 1 to 11 are of free type with other associated information. Module 12 is a fixed type dummy module with no connections. Modules 13 to 17 are identical and can be modelled as a single X-bar module with no pins though its sub-components have pins. The plug connector (pins 19-37) can be segmented as three fixed modules with 6 pins each.

4.2 NET INTERCONNECTION RULES

Three basic wiring rules are briefly mentioned.

Star connection: This connects all the pins in the net to the source (specified module-pin) directly.

Serial connection: Connects all the pins in a net in a serial fashion by creating a single string from the source pin to the final sink in an optimum manner.

Minimum distance: Connects all pins in the net in an optimum fashion without regard to source or sink or fan-out restrictions on a pin.

In addition to these above wiring rules the designer may specify any unique wiring rule in a free standing subroutine. Critical wirelengths can be monitored. Artificial nets may be added. The modules are also modelled very flexibly. Therefore by using any clever scheme almost any design problem can be modelled.

4.3 PROGRAM OVERVIEW

This is explained in the flow chart of Fig. 4.2.

The input data: This is data necessary to describe the model to the program. Consists of board dimension, number of nets, modules, wiring rules, module descriptions etc. and other program control parameter.

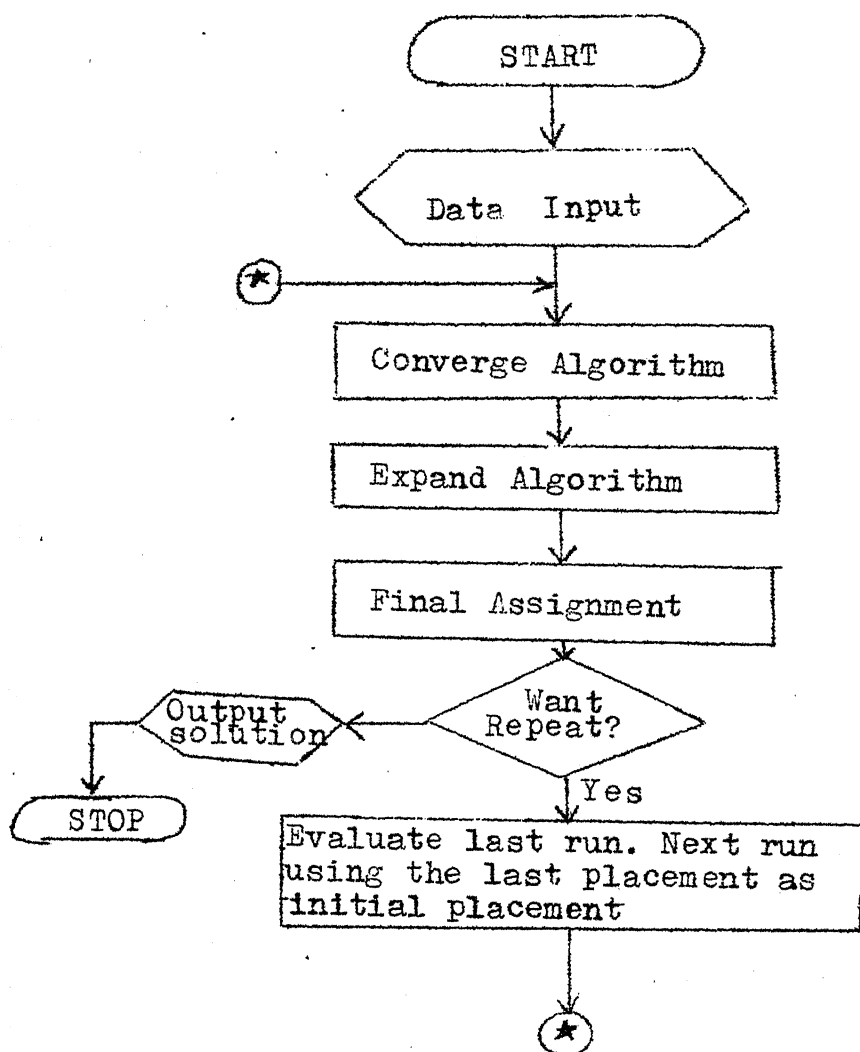


Fig. 4.2: Flow chart of Force Vector Method

'Converge' process: An iterative process that incrementally moves the modules to an oblique layout.

Modules are considered dimensionless points within a system of forces. Also reorders net interconnections between iterations in an attempt to optimize them based on wiring rule.

'Expand' process: This has not been implemented in the pascal program of this project. This process considers modules with true physical dimensions. Physical overlap creates a new type of force that acts to separate the components. Electrical attraction forces still apply. Boundaries of board contribute additional forces.

Final 'assignment': Necessary for the assignment of modules that must be mounted on predetermined sites, i.e. orderly rows and columns. Becoming increasingly important with use of ICs and automated production uses Hungarian assignment algorithm.

4.4 CONVERGE PROCESS

The aim of this process is two fold: An optimum orientation for all the modules with respect to each other and an optimum interconnection for each net. Optimality is with respect to the objective function of total line length. This iterative procedure has two steps in each iteration, which infact is a coordinate transformation of modules: Firstly all forces are computed for all modules in their current position, i.e., in the same state of the process. Secondly all modules are then relocated by a fractional amount of the resultant vector at the same time. The incremental approach eliminates dependency upon the

order of selection of modules for the move. The incremental fraction (i.e. 'Mobility Factor') is chosen to give speed of computation but at the same time it should not destroy the general orientation which was the basis to calculate this move.

4.4.1 VECTOR FORCES ON A MODULE AND SORT-DELETE TECHNIQUE OF COMPUTING RESULTANT

The process begins by computing the best interconnections for each net according to the desired wiring rule and the current module layout.

Each line (i.e., the rectilinear or 'Manhattan' distance) between a module and its connected neighbour is treated as a separate X vector and Y vector on the module.

Starting with the vector pairs of greatest magnitude (X and Y separately), pairs of vectors are deleted from the module. All remaining vectors (X and Y separately) on the module must be of same sign and they are added together into one vector. Figures 4.3 and 4.4 illustrate how this technique (named 'Sort-Delete Technique') operates and how it minimizes the rectilinear distance (which is what we seek) rather than the pythagorean distance.

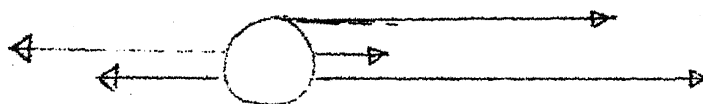


Fig. 4.3(a): Module with all X vectors equal to linear X distance to connected modules.



Fig. 4.3(b): One pair of vectors of opposite sign (greatest magnitude) deleted.

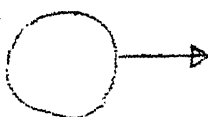


Fig. 4.3(c): Resultant vector

Fig. 4.3 Sort-Delete Technique

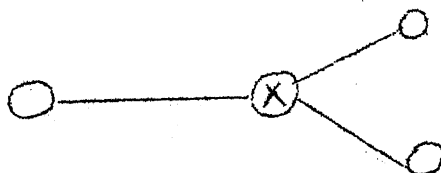


Fig. 4.4(a) Module X is in equilibrium for simple vector sum.

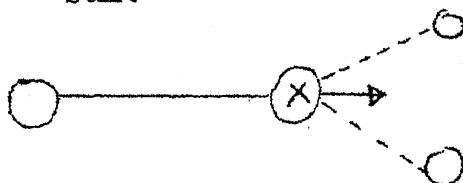


Fig. 4.4(b): Resultant vector in sort-delete technique

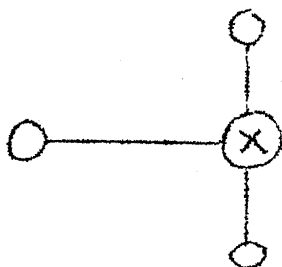


Fig. 4.4(c): Module X stable in Sort-delete technique - Rectilinear distance

Fig. 4.4 Reduction of Rectilinear distance in Sort-delete technique

4.4.2 REPULSION VECTORS

These forces are also to be considered acting on a module before computing the resultant vector on a module. This vector arises between the unconnected modules. These repulsion forces prevent the electrical attractions from pulling all the modules together in a central mass and also improve the stopping point by contributing to ensure that modules are on the proper side of each other at the end of converge process. This is illustrated in Fig. 4.5.

The amount of repulsion is an input variable. The paper suggests distance equal to two times typical module sizes. While this is retained in the implementation

it was found useful to drop the repulsion forces altogether once unconnected modules are separated by the same distance (twice typical module size).

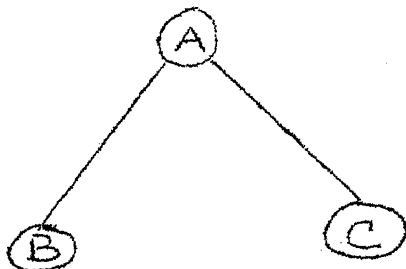


Fig. 4.5(a) Possible stopping point (B and C modules unconnected)

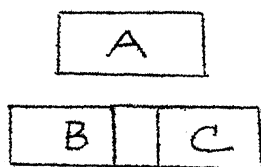


Fig. 4.5(b) Overlap between unconnected B and C at this stopping point.

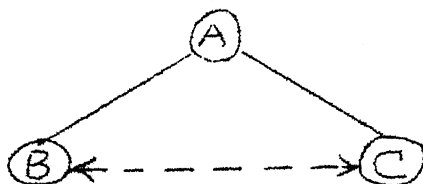


Fig. 4.5(c) Repulsions between B and C but still their length to A is the same.

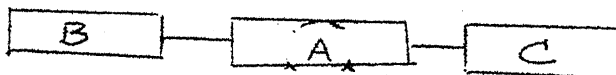


Fig. 4.5(d) Better stopping conditions with same minimum line length

4.4.3 MODULE TYPES AND VECTORS

The vector calculations must take into account the differences in modelling types. Vectors in X-direction and vectors in Y-direction must be treated separately for each combination of module types. Guide lines are:

- (a) Free modules: Normal calculations of vectors when the other module is a free module. If the other module is a fixed module then the free module receives twice the vector force.
- (b) Fixed modules: Receive no vector force.
- (c) X-bar and Y-bar modules: Treated as fixed modules. In the case of X-bar, the other reacting module will receive twice the force only in X-direction and no force in the Y-direction. The case is reverse for Y-bar.
- (d) X-Fixed and Y-Fixed module: Treated as fixed modules only as far as X-vectors and Y-vectors are concerned respectively.

4.4.4 MOVEMENT OF MODULES

Once the resultant vectors have been computed for all modules, the program adds the resultant vector times some scalar coefficient to the position of each module

which is its new position. This scalar coefficient is called the 'MOBILITY FACTOR' and is calculated automatically according to the formula:

$$\text{Mobility Factor} = \frac{1}{2} \frac{(\text{Average Vector})}{(\text{Maximum Vector})}$$

The one-half in the formula is due to the fact that vector computed between modules J and K is applied to both module J and module K and if no other module were included in the calculation, modules J and K will meet exactly half way between them in one iteration .

Secondly the formula should help in driving the solution to a stable finish for the following reason. At the beginning of the converge process, the placement may be very poor and vectors computed may be very large. If the maximum vector is large, the mobility factor will be smaller thus providing a desirable initial control. As the process continues and the forces decrease the mobility factor also increases as the vectors approach average value. This increase of mobility factor discourages the process from becoming slower as more iterations are completed.

4.4.5 NET INTERCONNECTION RULES AND ITS FREQUENCY OF IMPLEMENTATION

There is no way of determining the optimum order of interconnection for a particular net without considering all other connections on the board. In fact until a final placement is made there is no way of determining what effects all the modules and all the nets will have on a particular net interconnection. The wiring rules we implement for optimization apply only to any single state of this system of forces and moving modules.

In converge process, we can reorder (i.e. implement the wiring rules) the net between any two iterations. This however would be time consuming. It is apparent that between iterations involving very small changes in line length, no reordering would be necessary. However, when line length is changing sharply from one iteration to the next the reordering would be required. The program should therefore monitor this line length (which it does for determining the stopping conditions also as we shall soon discuss) and choose particular frequency of iterations after which it will reorder again. This comparison can be made on the first iteration after a reordering of net. This reordering frequency (in terms of iterations) is a matter of experiment and experience. The paper suggests the following function depicted in Fig. 4.6.

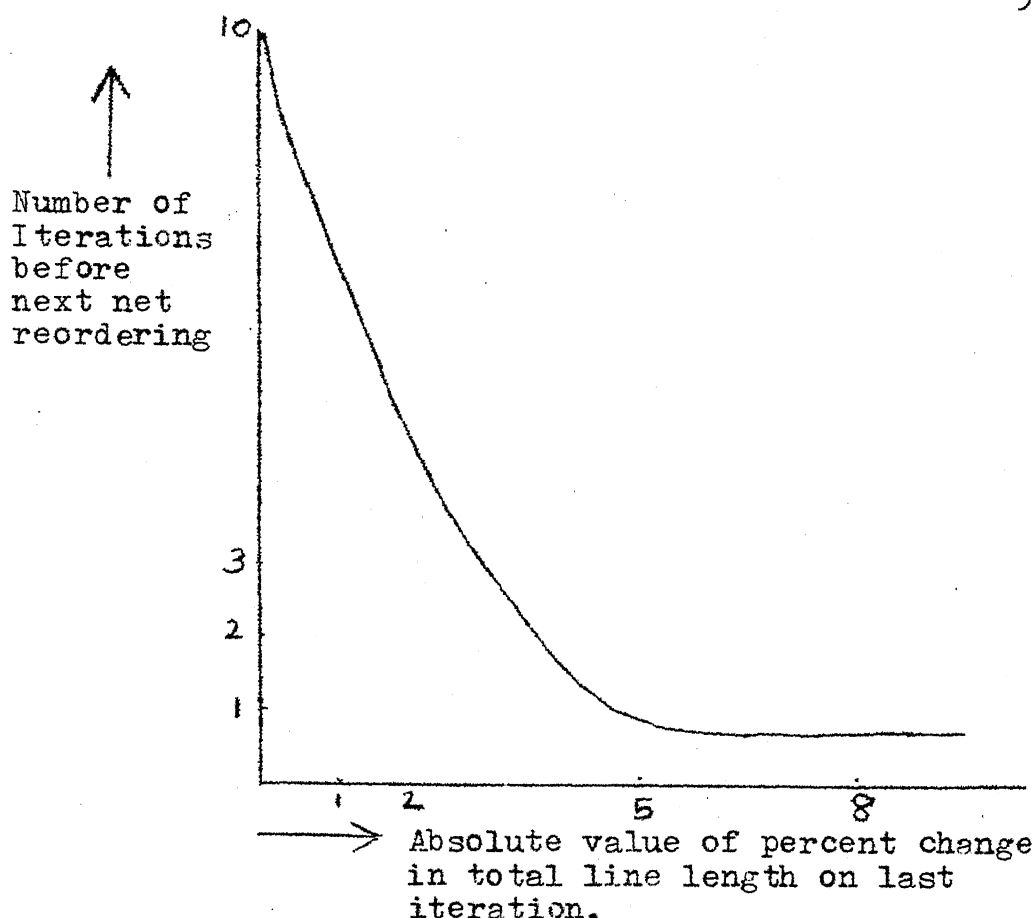


Fig. 4.6 Function for calculating net reordering frequency

4.4.6 PROGRESSIVE ADDITION OF NETS AND INTIAL CONDITIONING

Why progressive addition? The numerous combinations possible tend to add to the already numorous possible states that could be stable. Therefore, to help insure that the process approaches its stable state in the most unbiased increments possible nets are included progressively in the process.

Which nets first? A characteristic of most circuit designs is that most nets have just a pair of pins, or only three modules (pins) per net. Because there is only one way to connect pin pairs and that they also form the majority, it is essential that these nets be organized first. CONVERGE, therefore, begins by moving all modules to a stable state by considering only two pin nets. Once a stable state for forces generated with these nets is reached all nets with three or less pins are included and so on until all nets are included.

Initial conditioning: In an effort to further improve the unbiased starting condition of the problem all runs begin with the 'star' rule of interconnection regardless of their wiring rule to be implemented. But this is only for the First Pass through CONVERGE.

4.4.7 STOPPING CONDITION

This is a point of relative stability where total line length stops changing to within 'some small value' between iterations. This value again is a matter of experiment and experience. The paper suggests 10^{-7} as a typical figure for a 40 module problem. We may also put in the condition of predetermined number of iterations as stopping condition if such stability is not reached.

The program monitors line length during each iteration and uses a simple averaging scheme of line lengths from past iterations(after a reorder) to compare the change in the present iteration. This removes the restriction that the process must produce a non-increasing total line length and covers isolated cases of module oscillations.

4.5 EXPAND PROCESS

This process has not been implemented in this project for two reasons: that apart from complexity to the program & time factor, the experiments with few circuits could not indicate whether an infinite loop situation will be avoided for all circuit problems. Secondly where an assignment to predetermined slots are to be made after this, the few circuits tried showed little difference to the cost matrix whether the assignment is done directly after converge or after expand just allowing the modules to remove the overlap in converge. A brief description of this process follows.

In this process the modules should assume their appropriate physical dimensions and their motion is restricted by boundaries of the board. The principal function is to increase the sizes of the modules and

reduce the physical overlap between all modules without destroying the orientation prepared by the CONVERGE process. The paper suggests implementation of this process in two phases: 'bounded' and 'unbounded'. In the unbounded phase the modules are still not restricted by the boundaries of circuit board and physical overlap reduction is the main aim. In the next phase an additional vector force due to boundary of board is included.

4.5.1 REDUCING PHYSICAL OVERLAP

For the modules to overlap the distance separating them in both X and Y sense must be less than half the sum of the dimensions in either the X and Y sense. To separate this we depend on the vector component of repulsive forces which are based on overlap in both X and Y sense. Figure 4.7 below shows the physical overlap and the repulsion forces.

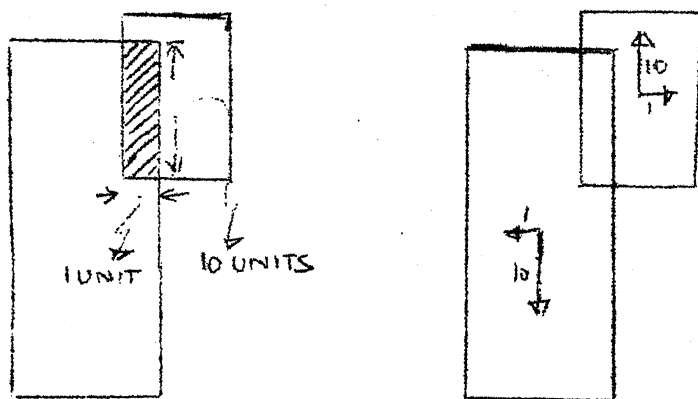


Fig. 4.7 Physical overlap and Repulsion vectors

If a pythagorean vector is used the movement would be effective. But use of only the smaller of X or Y components (in this case X) of repulsion would provide two advantages: Firstly overlap could be reduced more rapidly. Secondly the resulting conditions would be more accurate approximation of the original layout (after converge) and it is important to preserve the relative positioning derived by CONVERGE. Fig. 4.8 below illustrates this. When they overlap equally resultant vector sum can be used.

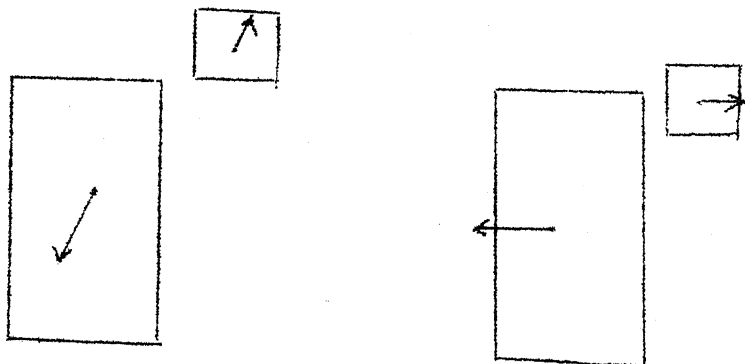


Fig. 4.8 Use of Resultant vector and smaller of X and Y vector

4.5.2 OTHER CONSIDERATIONS

Once these 'physical' repulsion vectors are computed for all modules, the 'electrical' attraction forces (due to net interconnection) or vectors as in converge are added to each module's total force vector. Certain constraints can be added to insure that overlap is reduced. It is

important to reorder nets between iterations but here we may use a simple constant (say 6) as reordering frequency because line length changes very little between iterations.

4.5.3 STOPPING CONDITION

Since the reduction in overlap is the main aim, that, and not the line length change, forms the criteria for stopping. Further, line length does not change much when certain value of minimum level of overlap is reached; (for example when overlap is less than 20 percent for the modules) the process then can be terminated.

4.6 FINAL ASSIGNMENT

The problem of assigning the modules to sites that were originally defined in the input data is different type of distance minimizing problem. At the end of converge/expand each of the 'n' modules has a cost (a_{ij}) of assignment (simply the absolute value of rectilinear distance to the site) to each of the 'n' defined sites. Each module has to be assigned to exactly one site and only one module is assigned to each site. If sites are extra, dummy modules can be assumed with no cost of assignment. A cost matrix whose elements are a_{ij} ($1 \leq i, j \leq n$) is presented to the assignment algorithm for deciding appropriate assignment

(without trying all the $n!$ possible assignments) such that $\sum_i a_{ip(i)}$ is minimum where p is a particular permutation of first n integers that corresponds to the assignment of sites to modules.

4.6.1 THE NATURE OF SOLUTION

It should be noted that instead of minimising interconnecting wirelength, the final assignment minimises the sum of the distances each module moves when assigned to a final site. In other words, the original criteria of electrical interconnecting lengths is ignored by this algorithm and hence its success depends on the approximation presented to it. That means the positioning at the end of converge/expand should be a good approximation of the final answer, causing only a small adjustment of movement.

4.6.2 HUNGARIAN METHOD

Basically the method successively modifies rows and columns of cost matrix until there is at least one zero component in each row and each column such that complete assignment can be made against n independent zeros in the altered matrix. This will be an optimal assignment in the corresponding positions in the original cost matrix since on the basis of Konig's theorem a constant

can be added to and subtracted from any row or any column without changing the set of optimal assignments (The matrix elements must be non-negative integers); and the theorem that if the minimum number of 'lines' required to 'cover' each zero element at least once is 'n' then there exist 'n' independent zeros in the matrix and vice versa.

4.6.3 STEPS OF THE HUNGARIAN ALGORITHM

We are given a cost matrix (elements a_{ij} are non-negative integers representing the absolute value of rectilinear distance of a module i to site j). Through the following steps we can find the 'n' assignments that will minimize the cost.

Step 1: If the minimum element in row i is not zero, then subtract this minimum element from each element in row i ($i = 1, 2, \dots, m$).

Step 2: If the minimum element in column j is not zero, then subtract this minimum element from each element in column j ($j=1, 2, \dots, m$).

Step 3: Examine rows successively, beginning with row 1, for a row with exactly one unmarked zero. If at least one exists, mark this zero with the symbol (Δ) to denote an assignment. Cross out (X) the other zeros in the

same column so additional assignments will not be made to that column (activity). Repeat the process until each row has no unmarked zeros or at least two unmarked zeros.

Step 4: Examine the columns successively for single, unmarked zeros and mark them with the symbol (Δ) to denote an assignment. Cross out (X) the other zeros in the same row so the corresponding resource will be assigned to other activities. Repeat the process until each column has no unmarked zeros or has at least two unmarked zeros.

Step 5: Repeat steps 3 and 4 successively (if necessary) until one of three things occurs:

- (a) Every row has an assignment (Δ).
- (b) There are at least two unmarked zeros in each row and each column.
- (c) There are no zeros left unmarked and a complete assignment has not been made.

Step 6: If (a) occurs, the assignment is complete and it is an optimal assignment. If (b) occurs, arbitrarily make an assignment (Δ) to one of the zeros and cross out (X) all of the zeros in the same row and column, and then go to step 3. If (c) occurs, go to step 7.

Step 7: Check (✓) all rows for which assignments (△) have not been made.

Step 8: Check columns not already checked which have a zero in checked rows.

Step 9: Check rows not already checked which have assignments in checked columns.

Step 10: Repeat steps 8 and 9 until the chain of checkings ends.

Step 11: Draw lines through all unchecked rows and through all checked columns. This will necessarily give the minimum number of lines needed to cover each zero at least one time.

Step 12: Examine the elements that do not have at least one line through them. Select the smallest of these and subtract it from every element in each row that contains at least one uncovered element. Add the same element to every element in each column that has a vertical line through it. Return to step 3.

CHAPTER 5

PLACEMENT IMPLEMENTATION

The force-vector heuristic technique discussed in Chapter 4 has been implemented in Pascal (Appendix II). The following is documented for easy understanding of the user.

This chapter will discuss data input, program overview and data output in the next three sections. Before discussing data input, it should be noted that the board dimensions are program constants and hence should be decided first and changed accordingly for each circuit. The difficulty in reading them from input file is because of 'window' and 'view port' dimensions which will also be program constants depending upon the Board dimension chosen. Therefore before executing the placement program for each circuit the board dimensions in X and Y directions and the window and view port dimensions are changed through the use of a Mic File. (Use of Mic file discussed in Sec 5.4).

Depending upon the circuit and routing requirement the X-span and Y-span of board are chosen as some integer units. Each integer unit may be typically $\frac{1}{10}$ " or 1 or 2 mm'. There is no restriction on the size (excepting window and

view porting constraints) as far as the placement problem is concerned. But if router program is to be used then the dimension is restricted to within 140x100 integer units.

Apart from these program constants of X-span, Y-span and the corresponding window and view port dimensions the following data are also to be supplied for each circuit for placement program.

5.1 DATA INPUT

This can be divided in three parts in the same sequence as follows. All measurements of length or size are in terms of the integer units decided for the board dimensions. Appendix I contains a listing of input for circuit 1.

Part 1 - Module data: The sequence of data under this part are:

1. Number of modules
2. Individual module data (modules are numbered in the order in which these sequence of data are read for each):
 - (a) Type of module (integer coding):
 - 1:Fixed
 - 2:Free
 - 3:X Fixed

4:Y Fixed

5:X Bar

6:Y Bar

- (b) Size in X-direction and then size in Y-direction from the coordinate position (centre) of the module. Normal coordinate conventions used.
- (c) Coordinate position in X and then in Y. Note that normal coordinate convention, i.e., origin at left bottom corner of board, is used.
- (d) Number of pins in the module (dual in-line).
- (e) Orientation of chip:
 - 0 : If not applicable
 - 1 : First pin on left top
 - 2 : First pin on right bottom

The data in (d) above and this are not applicable for bar modules (and hence = 0), but applicable to subcomponents of the bar module.

The subcomponents are identical. In circuit 1 example the ground and power lines are modelled as bar modules and have no subcomponents.

The data in (e) is not applicable for connector segment modules (and hence = 0). But data in (d) is applicable depending upon how many board terminal pins are modelled as one (fixed) connector segment module.

(f) Number of submodules:

Applicable only for those bar modules that contain subcomponents. (Hence = 0) for all other modules).

When this data is applicable, this must be immediately followed by data in (b) through (e) for each of the subcomponents.

Part 2 - Net Data: The sequence is as follows.

1. Number of Nets
2. Individual net data: (Nets are numbered in the order in which this data is read).
 - (a) Size of net (i.e. number of pins in net)
 - (b) Wiring rule for the net (integer code):
 - 1 : Star rule
 - 2 : Serial
 - 3 : Span (minimum spanning tree)

(c) For each pin in this net:

- i) Module number
- ii) Pin number
- iii) Sub-module number

Sub-module number is applicable only if the connection is to a subcomponent of a bar module whose number is given in (i). For all other cases it will be zero.

Note that the placement program does not move submodules. In this program submodule pin numbers are assigned before-hand to a net. They can be easily modified to be specified at the end of the placement. In any case the actual location of the pin is required for routing.

As far as the placement program is concerned, for calculation of vector forces in this net only the perpendicular distance to the concerned bar module will be taken into account.

It is important to ensure that, first, the nets read are nets of power line and ground line.

CENTRAL LIBRARY
Kandpur.
Acc. No. A 82563

Part 3 - Details for Terminal pins of Board: The

sequence under this is as follows:

1. Total number of terminal pins of board.
2. Length of a terminal pin. (The scale for this is the same as module size or position coordinates).
3. Orientation of Board terminals (integer code):
 - 1 : Terminals on top of board
 - 2 : Terminals to right of board.

It should be noted that if we are to use the router program with orientation 2 then it requires modification to one procedure (Power and Ground lines) in router program, which otherwise assumes orientation 1.

For placement program alone both can be used. This helps to adjust the window parameters so that the longer side is along X-axis.

4. X coordinate and Y coordinate positions (bottom point) of all terminal pins in the order which they are numbered on board.

5.2 DATA OUTPUT: (Refer Appendix III and Appendix VIII)

One part of this is tailored to use the router program. The board with the given dimension is thought of as a grid

of cells. The number of cells in the grid are equal to (X span*Y span). As mentioned in the beginning of this chapter it is restricted to (140x100) if router program is to be used.

Another part of the output is to produce the final placement configuration on the screen or plotter. This is done by procedure 'Draw Final Placement'.

Yet another part pertains to output statements which have been suppressed with comment delimiters. These are obvious on inspection of the program and can be made to output on a separate file. These are meant for taking a print out of various forces on modules, their resultant etc. and the cost matrix formed for assignment problem and the alterations in the matrix by the assignment algorithm. only that part of output pertaining to router program is discussed further. This in turn can be thought of as the following three parts in the same sequence.

Part 1 - Board Dimensions: The sequence is:

1. X-span of the board
2. Y-span of the board

These pertain to the board dimensions discussed in the very beginning of this chapter and are constants of placement program for each circuit. They are merely

presented at the output again for use by router program, routing for the same circuit.

Part 2 - Cell Numbers of all pins: The sequence is:

1. Sequence of cell numbers corresponding to all pin locations of all modules (including subcomponent pin locations of bar modules).
2. Integer 99998 to indicate that part 1 and part 2 (of input to router program) are terminated.

The formula for knowing the cell number of a pin location is $((Y \text{ coordinate position of pin}) * X \text{ span}) + (X \text{ coordinate position of pin})$. The cell number will be an integer since centre of a module is an integer position and its size is chosen as an integer depending upon the number of pins. Therefore each pin location is an integer position in X and Y. It is important to note that the choice of coordinate position in placement program makes the left bottom most cell as cell No. 1. But the router program has left top most cell as first cell increasing linearly. Therefore a conversion is required in coordinate position of pin before giving the correct cell number to router. This is automatically done in placement program.

The order in which the sequence of cell numbers are read is unimportant since they are required by router only

for the sake of marking these cells as occupied by pins and to physically define a module. However for convenience an order is adopted in this output inplacement.

Modules are examined one by one in the order in which they were input and all pin locations for this are output. When a bar module is encountered, if it has submodules then all the pin cells of all its subcomponents are output; if it has no subcomponent (for example, a power or ground line) then no cell number is output).

As soon as the first connector segment module is encountered the cell numbers of all board terminals are output.

Part 3 - Cell numbers of pins of each net: The sequence is:

1. Sequence of cell numbers to be connected to all pins of power line followed by integer 0 to indicate the termination of cells in this net (net 1).
2. Similar sequence for Net 2 (ground line) followed by integer 0.
3. Similar sequence for Net 3, Net 4 and soon till the last net. Each sequence is followed by integer 0.
4. Last net sequence is followed by integer 0 and then integer 99999 to indicate end of data under this part.

The order of the cell numbers in each sequence for a net depends upon the order in which they are stored in the linked list for that net in the placement program. Both this order and the order in which the net (i.e. the sequence of cells) are considered for routing may be changed by router program.

Changing the order within a sequence of cell numbers corresponds to implementing a wiring rule. Placement program can be modified to do that after final placement and output cell numbers. But that would alter the router program. The router implements some wiring rule by itself and a few more can be added to it.

5.3 PROGRAM OVERVIEW

5.3.1 STEPS OF THE PROCESS

Step 1: Initialise the device, window and view port. The window and view port dimensions are fixed by the 'init procedure'.

Step 2: This is done by procedure 'Readin', in the following sequence :

- (a) Read module data from input file (Ref. Sec. 5.1).

The details of each module is stored in the variable array 'module' including Bar Modules as 'component' records. If the bar module has any subcomponent

the details of all its subcomponent are stored in the variable array 'Barnap' as 'Bar component' records.

- (b) Fix the average size of the modules. This is required for calculating repulsion force.
- (c) The candidate modules and candidate sites are fixed for the assignment problem. The candidate modules are of free type or X-fixed or Y-fixed type. These module numbers are stored in the variable array 'Matmods' and their X and Y positions are stored in the variable array 'Mat-sites', i.e., the candidate sites. The Rectilinear distance of each candidate module to each of the candidate sites as at the end of placement problem decides the assignment matrix.
- (d) Read Net data from input file. (Ref. Sec. 5.1). For each of the net read in, form a linked list. Other details of the net are stored in the variable array 'net' as a 'net-cell'. Each net-cell has a pointer to the appropriate linked-list.
- (e) Note down the number of pins in the net with maximum pins against variable 'maxpin net'. This is used for progressive net addition.

(f) Read the terminal pins data (Ref. Sec. 5.1). Store the X and Y positions of each of the terminals pins of the board in the variable array 'Termpinlocs'.

Step 3: Implement the procedure 'converge' discussed in detail in Chapter 4. This is done by procedure 'converge' which calculates the vector forces on each module and moves them as discussed in Chapter 4, by mainly using its two local procedures 'Net Reorder' and 'Move'. The procedure 'Net Reorder' implements the specified wiring rule for each net by deciding which module (pin) is to be connected to which module (pin). It then updates an important array variable 'Temp' that stores in its records all the connection details. The procedure 'Move' then calculates all the vectors on each module using 'Temp' variable and also the resultant and mobility factor in X and Y directions separately. Next it adds the change in position to the appropriate field in each component record in the array 'Module'. 'Converge' procedure is illustrated in Fig. 5.1.

Step 4: The assignment process discussed in detail in Chapter 4 is implemented by the procedure 'assignment' mainly through two procedures local to it: The procedure 'Fix-matrix' forms the cost matrix for the assignment problem (Ref. Step 2(c)). It should be noted that in the

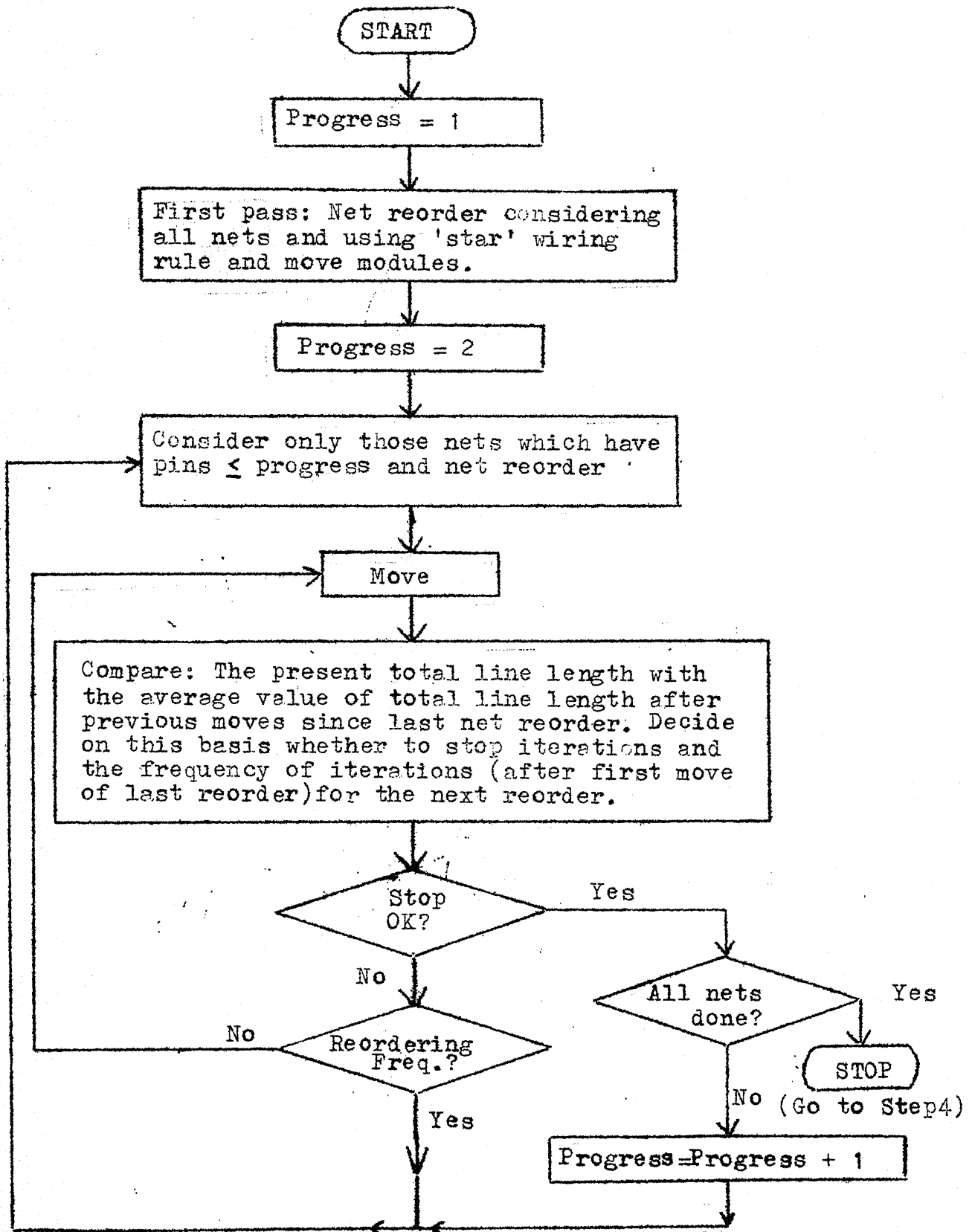


Fig. 5.1: Flow chart for Step 3 (converge)

case of X-fixed (and Y-fixed) modules the cost of assigning it to any site not having the same X-coordinate (Y-coordinate) is made arbitrarily large. The procedure 'Hungarian' then takes over and implements the steps 1 through step 12 of the Hungarian assignment algorithm discussed in Section 2.3 by means of a number of procedures local to it.

Step 5: Draw the final placement configuration on the screen/plotter. This is done by procedure 'Draw placement' which invokes it two local procedures 'Mark pins' and 'Mark term1s' and also to procedure 'Draw Rect'. All the modules (except subcomponents of Bar modules and connecting segments) are drawn in their final positions on the board and numbered. The pins of modules and the terminal pins of the board are drawn and their first pin is numbered.

Step 6: Output the program constants X-span and Y-span of the board for use by router program.

Step 7: Examine modules one by one. Output cell numbers of all pins (Ref. Sec. 5.2) of all modules considering the first cell to be on left top corner. This is used by router program. No pin-cell number output for bar modules with no subcomponent. When the first connector segment is encountered the cell numbers of all terminal pins is output by accessing the variable 'Term pin locs'.

This step is implemented by procedure 'Write all pin cells'.

Step 8: This is the last step in placement program, implemented by procedure 'write net pin cells'. Examine nets one by one and output cell numbers of all pins in each net (Ref. Sec. 5.2).

5.3.2 SOME IMPORTANT VARIABLES AND CONSTANTS

Avlength: Global. Gives the average value of total line length after moves (excluding the present move) since the last net reorder. Ref. Section 5.3.1, Step 3 flow-chart.

Avsize: Global. Gives average size of modules. Used for calculating repulsion force. Ref. Section 5.3.1 Step 2(e).

AVX(MX and MOBFAC TX): Real variables local to procedure 'Move'. Once the resultant force on each module is calculated, the average force on modules, maximum force on modules and the mobility factor on modules are calculated against these variables (in X direction). Similarly for AVY, MY and Mob Facy.

Bar Map (Bar comp and Bar ind): Global. Illustrated below. The upper bound of 20 for the array can be varied as required.

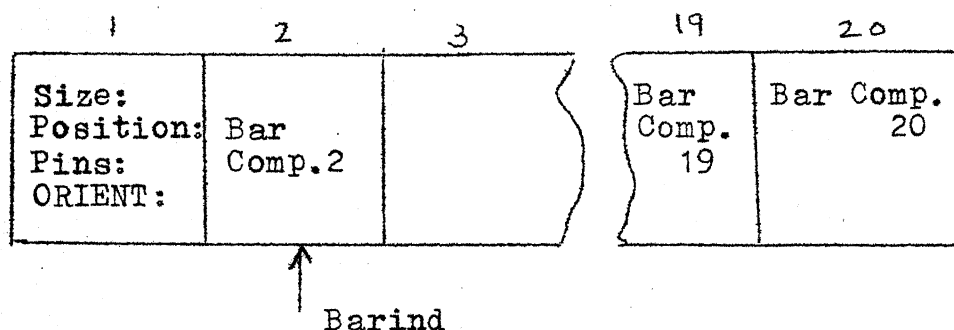


Fig. 5.2

BIT: Local to procedure 'force'. A Boolean array to indicate whether a module is connected or not to each of the other modules. Updated every time a particular module is considered.

DIFFY (ORIENT and PINLEN): Global, but used only in procedures 'Markpins' and 'Write-cells'. Pertains to distance between pins in a module and should be assigned a proper value in these procedures by the user. 'ORIENT' refers to orientation of chip (Ref. 5.1 part 1(e)). 'PINLEN' is a constant local to procedure 'Mark pins' to decide the length of pins. User specified.

FX(and FY): Array variable local to procedure 'force' and used for storing all the forces on a module updated for each module. This array is passed on as parameter to function resultant which calculates the resultant force in X-direction (and Y-direction) on the module by sorting them

and deleting pairs of opposite sign as discussed in Chapter 4. The same identifiers are used as real variables local to procedure move, in which case, it is merely used as a temporary storage to aid in calculation of maximum force in X direction (and Y direction) on modules.

Freq.: Global variable whose value is decided by procedure 'compare'. This integer value is the frequency of iterations of converge between the net reordering.

From, UPTO(and Map): Array variables local to procedure 'Net reorder'. 'From' and 'Upto' give indices to 'Map' which stores module numbers which are to be connected in a net updated for each net considered.

INC: Global. Boolean variable to decide whether a particular net is to be included for consideration during progressive net addition of the 'converge' process.

Matrix, Mat Mods and Mat sites: Matmods and matsites are global arrays discussed in Sec. 5.3.1, step (c). 'Matrix' is the cost matrix formed from these for the assignment problem. A row of the matrix pertains to distance of a module to all the sites and other information (whether the element is unmarked crossed or assigned during the process of assignment i.e. 'Daagtyp'). Also

TERMLTYP: Specifies orientation of board terminals.

Refer Section 5.1, Part 3.

TEMP: An important array variable that stores all interconnection data between modules. 'Tempmax' can be changed by user. Each module has a pointer to one cell of 'Temp' as illustrated below.

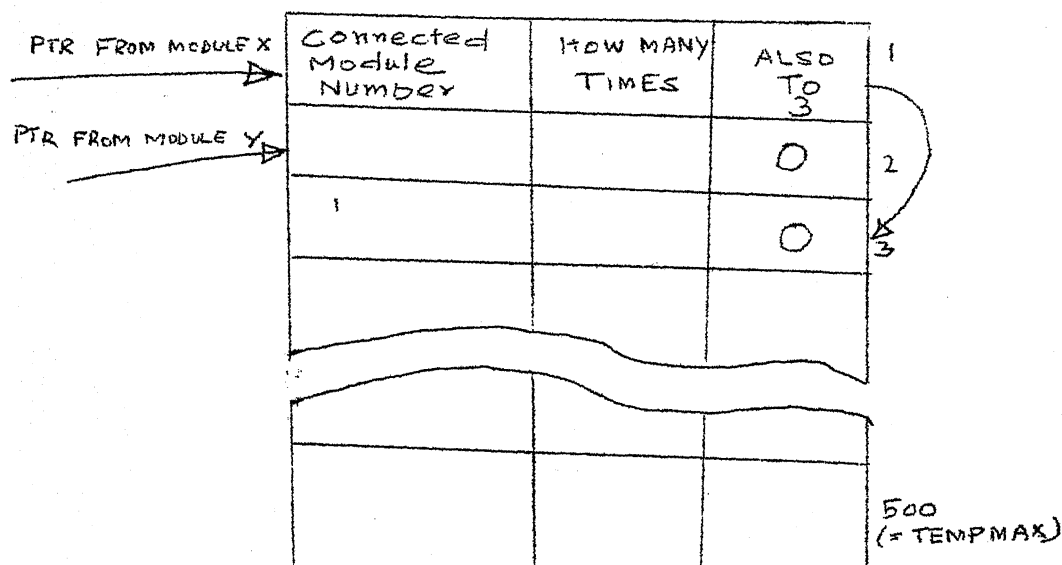


Fig. 5.5

TERMLOCS: Array storing positions of all board terminals.

5.3.3 BRIEF NOTES ON PROCEDURES

Most of the procedures have been outlined in Section 5.3.1. A further note is added here.

Static Nesting of Procedures and Functions:

MAIN

```
DRAWRECT
CONVERGE
  READIN
    READMODULE
    FIXAVSIZE
    FIX MATMODS AND SITES
    READNETS
    FIXMAXPIN
    READTERMLS
  DISTANCE
  LINELENGTH
  NETREORDER
    STARREORDER
    SERIAL REORDER
    SPANREORDER
  INSERT
  MOVE
    FORCE
      RESULTANT
      SIGN
      XVECTOR
      YVECTOR
      XREPULSE
      YREPULSE
    ROUND
  COMPARE
  ASSIGNMENT
    FIXMATRIX
    HUNGARIAN
      FIXSTATUS
      MATPRINT
      STATPRINT
      STATCPRINT
      ZEROPRINT
      ROWMIN
      ASSROW
      ASSCOL
      CONDITIONCHECK
      REDUCE MATRIX
        TICKLINES
        COVERLINES
        DO FINAL MAT
      ASSIGN
  WRITE CELL
  WRITE BAR CELL
  DRAW PLACEMENT
    MARK PINS
    MARK TERMLS
  WRITE ALL PIN CELLS
  WRITE NET PIN CELLS
```

Fig. 5.6: Nesting of Procedures.

Procedure Net Reorder: Refer Sec. 5.3.1, Step 3. This implements three wiring rules. In 'star' wiring rule the module in the central position of the net (or a Bar module without subcomponents) is connected to all other modules in the net. 'Span' Reorder implements a minimum spanning tree. In Serial Reorder every vertex has an incidence of atmost 2. In all, if a net has 'N' pins then there are (N-1) edges to be specified using 'From' and 'Upto' array. At the end of net reorder the procedure 'Insert' updates the array variable 'Temp'.

Procedure Move: Examines each module. From 'Temp' array information calculates attraction force to each connected module and repulsion force to each unconnected module specified by 'Bit' array. The functions X vector, Y vector, X repulse and Y repulse calculate these forces with due regard to module types. The local procedure 'force' uses these functions and the function 'resultant' to update the Force field on each module. Once this is done 'Move' calculates the mobility factor and updates the position field of individual modules.

Function Distance: Given two modules as parameter, this calculates the rectilinear distance between them with due regard to module types.

Procedure line length: Uses above function and connection details in 'Temp' array. Returns the total line length for the current lay-out and net reorder implementation.

Procedure compare: Compares line length after every move with the average line length in previous moves since last net reorder. Depending upon the absolute value of the percent change in this comparison it decides to stop the iteration or the frequency of iterations for the next reorder.

Some Local procedures of Assignment: The procedure 'Fix Matrix' forms the cost matrix for the assignment problem. 'Fix status' initialises the matrix. The 'print' procedures are added for diagnostic purposes and have been suppressed.

Procedure Hungarian: This implements the assignment algorithm of Section 4.6.3. The procedures 'Row Min' 'colmin' correspond to steps 1 and 2 of algorithm. The 'while' loop containing procedures 'Ass Row', 'Ass Col' and 'Condition Check' correspond to steps 3, 4 and 5 and 6 of algorithm. While within the loop it is analogous to out come in step 5(b), If condition check causes exist from while loop due to success it means the out come in step 5(c). If the out come corresponding to step 5(c) is caused

then the procedure 'Reduce Matrix' takes over, which corresponds to step 7. The repeat loop in this case ensures that after step 12, we return to step 3 of algorithm. The inner procedures represent steps 8 to 12 of the algorithm. The first repeat loop within 'tick lines' implements steps 8-10 in the form of two inner 'for' loops and an inner 'repeat' loop respectively. Procedures 'coverlines' and 'Do final mat' implement steps 11 and 12 of algorithm.

Procedure Assign: In case of success this procedure updates the position field of individual modules.

Procedure Draw Rect: Draws all the modules except subcomponents of Bar modules called by converge process to give a visual picture of converge to the user. Procedure draw placement also makes a call to this.

Procedures to Output Cell Numbers: It should be noted that connector segment modules (i.e., those containing board terminals) are identified by their 'orient' field = 0 but pins $\neq 0$ whereas bar modules are identified by orient = 0 and pins = 0. In case a bar module has subcomponents its barsubs $\neq 0$ and this field is zero for all other modules. The procedure 'write cell' outputs a particular cell number when the module number and pin

number (may be a board terminal number) are specified. If pin number is specified as zero in the case of a connector segment then it outputs cell numbers of all the board terminals. The procedure 'write bar-cell' outputs the cell number of a particular pin of a subcomponent when the bar module, its subcomponent number and pin of subcomponent are specified. If the last two parameters are zero then it outputs cell numbers of pins of all the subcomponents of that bar module. These two procedures are invoked by 'write all pin cells' and 'write net pin cells' to produce output data as discussed in Section 5.2.

5.4 USE OF 'MIC' FILE FOR PROGRAM EXECUTION

As seen in Section 5.1, for each circuit the program constants of window and view port dimensions are to be changed depending upon board dimensions. The board dimensions are also kept as program constants for this reason.

Further when the placement program router program and the plotting routine are to be done in sequence or independently and that for many circuits, extensive renaming of files will be required to avoid input output files confusion. For this reason as soon as the program is executed the input file is stored back as PLA1.IN (i.e. for circuit 1; For circuit 2 it is PLA2.IN etc) and the placement output as PLA2.OUT etc.

These require a series of instructions for execution and are convenient to do through the use of a 'MIC' file. For each circuit a Mic file instruction is necessary. For example, the 'Place1.MIC' file instructions require a simple command 'DO PLACE1' to carry out necessary instruction for placement problem of circuit 1. 'Place2' for circuit 2 etc.).

This can be immediately followed by MIC file instructions 'Route1' and 'Plot1' and then 'Name1' instruction which renames the input/output files as they should exist. If the user wishes to terminate his work with placement or in any other combination 'Name1' must follow 'Place1'.

The Mic file instructions for circuit 1 are enclosed as Appendix VII.

CHAPTER 6

ROUTER IMPLEMENTATION

The router presented here (Appendix IV) is an existing implementation [10]. This chapter merely attempts to document it more clearly for ease of user and to provide an environment integrated with the placement program.

6.1 DATA INPUT

This can be thought of as a sequence of four parts. The first three parts and the sequence within each part is identical to that of placement program output. Since the input data can be given independent of placement, the sequence is repeated again and points pertaining to router program alone are briefly discussed.

Part 1 - Board dimensions:

1. X span of board
2. Y span of board

These are read against router program variables 'X Coord' and 'Y Coord'. Program considers the board as a grid with (X span * Y span) cells limited to 140x100. This can be changed depending on memory available. A cell is a $\frac{1}{10}$ " square or 1 or 2 mm square depending on circuit design

typically. The first cell is assumed at top-left corner increasing towards right and then on second line and so on. The last cell is at bottom-right corner.

Part 2: The sequence again is:

1. Cell numbers of all pins of all modules
2. Integer 99998.

The order in which the cell numbers are read is unimportant as they only physically define a module. The router program recognises only cell numbers without any regard to type of module.

Part 3 - Cell numbers of pins of each net: The sequence is:

1. Cells of all pins connected to ground line followed by integer zero.
2. Cells of all pins connected to power line followed by integer zero.
3. Cells of pins in each of all the other nets. Each net is a sequence of cell numbers and followed by integer zero.
4. Integer 99999.

The order in which each sequence under (3) above is considered may be changed by the router program by means

of either of two procedures 'Area ordering' and 'Mid ordering'. The order in which pairs of cells are considered within each sequence may also be changed by router program by means of function 'nearest', which infact is an implementation of minimum spanning tree wiring rule.

The Nets 1 and 2 are implemented as a 'star' wiring rule in which each of the pins is connected to the ground line or power line as the case may be.

Part 4 - Character codes as data: The program expects the following sequence of single characters as code for choosing methods of routing:

1. Character 'M' or any other character: If 'M' is read the nets are ordered for routing depending upon which net (any pin of it, i.e. any cell within the sequence of cell numbers) is farthest from centre of board by invoking procedure 'Mid order'. If M is not read then the nets are ordered for routing depending upon which covers most area of board by invoking procedure 'area ordering'.
2. Character 'B' or any other character: If 'B' is read then the ordered nets are considered in descending order (of distance or area covered) for routing. If 'B' is not read then they are considered in ascending order.

3. Character 'C' or any other character: If 'C' is read then 'chaining method' is used for finding a path between given pairs of pins. If 'C' is not read then 'Steiner's tree' (i.e. procedure 'Straner's tree') is used for the same purpose.
4. Character 'O' or any other character: If 'O' is read then the program does not route for the other side. If 'O' is not read routing is done on both sides.

6.2 DATA OUTPUT

The router program operates on the input data to produce the following data strictly in the same order. The actual drawing on the screen or plotter is not done by the router program but by a small plotting routine in fortran which is discussed in Section 6.3. This data again can be considered in four parts in sequence. An example is appended. (Appendices V and VI).

Part 1 - Data to Ground and Power Lines: For each line 3 points (cells) are output including the corner point. Though line drawn covers only one cell in width, it is actually two cells in width. The sequence is:

1. Three cell numbers which are to be joined by two straight lines in the same sequence followed by integer 0 as delimiter.

2. Three cell numbers similar to above, but which make the power line.
3. Integer 99996 to terminate part 1 data.

Part 2 - Routing Data for side one: This mainly has many sequences of cell numbers. The cell numbers in each sequence are to be connected by straight lines in the same order by plotting routine to make the routing. Each cell number in one sequence does not necessarily denote a pin location, but at least one and at most two of these cells must correspond to pin locations, since these sequences are formed by either of two procedures ('chaining method' or 'straner's tree') which receive a cell pair as their parameter. Hence sequence is:

1. Sequences of cell numbers, each sequence followed by integer 0 as delimiter.
2. Integer 99997 to terminate data in this part.

Part 3 - Routing Data for side two: This is similar to part 2. But produced only if desired by user through the character code at input as discussed in Section 6.1. This part, whether output or not, is terminated by integer 99999.

Part 4 - Data of unmade connections: This part is generated on a separate file called 'out' since it does not concern the plotting routine. This lists out in sequence for each unmade connections:

1. Net number
2. Pin number in the net

These connections have to be manually made. An example of router output is included as Appendix V.

6.3 PLOTTING ROUTINE: (See Appendix VI and IX)

Though the router program produces a sequence of cell numbers to be actually connected, it does not do the actual plotting or drawing. This is done by a small Fortran Routine using just 'line' cells of GPGS.

This routine uses two data files which are nothing but the output file of placement program i.e., same as input file of router program and the output file of router program. These two are renamed FOR 20.DAT and FOR 21.DAT respectively.

FOR 20.DAT is only partially used, i.e. to mark short dashes at cell positions corresponding to all pins on the circuit board, i.e. the cell numbers after integer 99998 in the file are not used.

FOR 21.DAT is used as follows:

- (a) For drawing power and ground lines, i.e. cell numbers upto integer 99996.
- (b) For drawing all possible wire connections on first side, i.e. data between integers 99996 and 99997.
- (c) For drawing interconnection on the second side, i.e. data between integer 99997 and 99999.

Before drawing connections on both sides the program awaits for user response (type any integer) and also for clearing device/plotter.

6.4 PROGRAM OVER VIEW

6.4.1 STEPS OF THE PROCESS

Step 1: Read X span and Y span of the board.

Step 2: Read cell numbers of all pins of all modules from input file (Ref. Sec. 6.1, Part 2). Update the information for these cells as follows: side one occupied and the cell not being used by any particular path. Say, the fields for each cell used for this purpose are:

Cell Info [Cell number].side = 1

Cell Info [Cell number].Path code = 1

One more field i.e., cell info [cell number].Link cell is not of concern at this step.

Step 3: Read cell numbers of pins of all nets (Ref. Sec. 6.1, part 3). Form a 'net matrix'. Each row represents one net and each element of this matrix has two fields: Cell number corresponding to the pin and Boolean info whether the cell has been tried for routing by the program (i.e. 0 or 1 for 'untried' and 'tried' respectively). Store the cell numbers of pins in the net in the appropriate element and update the other field to zero (i.e. Not yet tried for routing. The elements of this net matrix are accessed as net matrix [Net No., Pin No.] where net number is the order in which the nets were readin and pin number is the order in which the pin (i.e., its cell number) was readin. The end of pins in a row is indicated by a zero stored in the cell number field of the element.

Step 4: Note down the total number of nets.

Step 5: For all border cells update cell info as in step 2. This is done by procedure 'initialization'.

Step 6: Demarcate power and ground lines. All second and third cells from left boundary and second and third cells below top boundary upto a distance (in this case 16 cells from mid point) are encoded as in step 2 but here the PATH code = 2, indicating that they are to be used for connections of power net (i.e. actual net number one).

Similarly ground line uses second and third cells at right and below top boundary with PATH code = 3. This step is implemented by procedure 'POLINE'.

Step 7: Output three cell numbers (i.e., end cells and corner cell) of power line followed by integer 0 and then three cell numbers of ground line followed by integer 99996.

Step 8: Perform the following steps for side one (Program Variable 'Side' = True).

Step 9: Consider 'PATH' Two; i.e., connections to power line. Examine each element of first row of net matrix. Treat each cell number as 'start cell' and invoke procedure 'Straners tree' with 'Start cell' and 'end cell' as parameters every time. Here the 'end cell' can be taken as 2 for parameter's sake. No need to change end cell's path code since this is already fixed in procedure POLINE. 'Straner's tree' implements Lee's Algorithm and finds a path between the start cell specified and any of those cells with the same PATH code as Path under consideration. When a Path is found it outputs a series of cell numbers to be interconnected by straight line for this path (Refer Sec. 6.1; part 2).

Step 10: Consider 'Path' Three. This is similar to step 9 but connections to ground line are made examining the second row of matrix (i.e. actual net number two). with 3 as the path code for cells and as end cell parameter. The output in this step is similar to step 9. Steps 9 and 10 are in effect implementation of star wiring rule for the first two nets.

Step 11: Read character code and order the remaining nets (i.e., net 3 onwards). By invoking procedure 'Mid-ordering' if character 'M' is read or procedure 'Area ordering' otherwise. In the case of 'Mid ordering' the nets are ordered depending upon the maximum (Rectilinear) distance (for any pin of a net) covered by the net from the mid point of the board. In the case of 'Area ordering' the nets are ordered on the basis of board area covered by each net (i.e. greatest X span covered * greatest Y span covered by the net). The net number and its associated area or distance is stored in the ordered sequence in a 'Net order array'.

Step 12: Read two more character codes. If the first character read is 'B' then the 'net order array' will be examined in descending order. Otherwise in ascending order after sorting. If the second character read is 'C' then the procedure 'chaining method' will be used for finding path between cell pairs; otherwise Steiner's tree method

will be used. 'Chaining method' is similar to 'Straners tree' except that the path has to end at the cell number specified as 'end cell' instead of at any cell having a same 'path code' as the path under consideration. Both require 'start cell' and 'end cell' parameters with the path number (all connections in one net have same path number) specified. While routing on first side, both store start cell, end cell and path number information in a 'second side' array (along with the net number and pin number corresponding to start cell) if a path can not be found.

Step 13: Repeat the following for each net in the 'net matrix' in the order decided by the 'net order array' which yields a 'row number' (net number) of the net matrix:

- (a) In the selected row, choose the cell number of the first element (i.e., first pin of this net) as start cell. Updated the 'tried' field of this element to 1.
- (b) Store the pin number of the tried pin in this net in a 'pin number array' sequentially.
- (c) Repeat the following until all pins of this net have been tried:

- i) Implement a minimum spanning tree starting with this first pin. This is done by function 'nearest' using a 'local array'.
- ii) Each time a new vertex is added to the spanning tree take that cell number in the net as 'starting cell' and find the pin number (from 'pin number array' which stores pins already tried for routing) nearest to it and take that cell as 'End cell'.
- iii) Store the start cell's pin number in the 'pin number array' and update the 'tried' field in the corresponding element to 1.
- iv) Invoke 'Straners tree' or 'chaining method' with these two parameters.

The program assumes path number 5 for the first net routed in step 13 and increments this path number for every subsequent net routing tried. (Path number 4 also can be assumed in the beginning).

Step 14: Perform the following steps for side two. (Program variable 'side' = false); output the integer 99997 and initialise the 'second side' array index' to 1.

Step 15: Read character code. If the character read is '0' then no routing on second side, else proceed with routing on second side. If no routing, then invoke procedure 'No lay' with 'second side array pointer' as parameter, incrementing it by three after every invocation since every third element of this array contains information on the start cell that was not connected (if any). If routing is to be done then invoke 'Straner's tree' or 'Chaining method' with 'Start cell' and 'End cell' information contained in this array (if any) as parameters till such pairs are exhausted. At this time if 'Straner's tree' or 'Chaining method' is not able to find a path it will invoke procedure 'No lay'. The procedure 'No lay' writes out (or 'out' file) the start cell, i.e., the pin numbers and net numbers which has to be manually connected. Finally output the integer 99999.

6.4.2 SOME IMPORTANT VARIABLES AND CONSTANTS

X Coord and Y Coord: These are the board dimensions. Ref. Sec. 6.1, part 1. Limited to 140x100.

Array 'A': This is the 'net matrix' array discussed in step 3 of Sec. 6.4.1. The indices of the array can be changed by user to suit a circuit. The 'select' and 'data' fields of the elements of this matrix correspond to the 'tried' and 'cell number' fields in our discussion.

AC: An integer var parameter used in procedures 'Straner's tree' and 'Chaining method' and indexes the element (in a list used by these procedures) that contains No. of the target cell (i.e., 'end cell' or a cell having the desired 'path code').

B: This corresponds to 'net order array' discussed in step 11 of Sec. 6.4.1. The fields of the elements of this array are self explanatory except the 'order' field, which corresponds to 'net number' field in our discussion.

There is also ^{one more} array B local to function 'nearest' mentioned as 'local array' in step 13 (c)(i) of Sec. 6.4.1. This is further explained under notes on procedures in the next subsection. The index bounds can be changed by user.

Cell Status: Corresponds to 'cell info' array mentioned in step 2 of Sec. 6.4.1. The fields 'select', 'pin' and 'chain coord' respectively of each of its elements correspond to the fields 'side', 'path code' and 'link cell' mentioned in the same step. Index upper bound is limited to 14000.

ISig: This is an index to the 'net order array' (i.e. array 'B') that gives the net number to be considered for routing.

JPOS: This is an index to get the pin number (in the selected net) to be considered for routing as start cell.

LISTL: An array for use by 'Straner's tree' or chaining method to store all cell numbers available for laying a path between the start and end cells. (or the target cell). Index bounds can be changed by user.

Sec Side: This index bound also can be changed by user. Array meant to store the start cell (&corresponding net number and pin number), end cell and path number when the procedures fail to find a path on the first side between start and end cells.

TREE: This is the ('tried' or connected) 'pin number array' pertaining to the net under consideration. Discussed in step 13(c) of Sec. 6.4.1.

6.4.3 BRIEF NOTES ON PROCEDURES

Static Nesting Sequence of Procedures:

```

MAIN
  NOLAY
  STRANERTREE
    NEIGHBOURHOOD
      CHECK
      BACKTRACK
      PATH
  CHAINING METHOD
    NEIGHBOURHOOD
      CHECK
      BACKTRACK
      PATH
  INITIALIZATION
  POLINE
  MIDORDERING
  AREA ORDERING
  NEAREST

```

Fig. 6.1: Nesting of Procedures

Procedure 'STRANERSTREE' (and 'CHAINING METHOD'): This has been briefly outlined in Sec. 6.4.1 step 9 and 12. Therefore this paragraph only reviews the procedures local to these two procedures which are also similar. The procedure 'Neighbourhood' begins with the 'start cell' stored in the variable 'LISTL', and fills up the list with all the available cells for laying path between it and the end cell (or target cell) by means of procedure 'Check'. Procedure check passes a cell for inclusion in the list if its 'path code' is free (i.e. = 0) or has the same as that of 'end cell'. The second condition is important if connection is on second side. All four neighbouring cells of each new cell added in the list is checked in this manner. Then by means of two list pointers, function 'path' checks whether a path exists. This returns True only if some cell added in the list has the same path code as the path number under consideration in the case of 'Straner's tree' and if the cell number is same as that of end cell in the case 'chaining method'. Also each cell added in the list has its 'link cell' field updated except the start cell for which this field is zero. If a path exists the procedure 'Back track' traverses the cells in the list by means of 'link cell' field (i.e. 'chain coord' field in program) starting from the end cell or target cell. It outputs the cell numbers when-ever a change of direction is involved.

PROCEDURE 'NEAREST': This takes unconnected ('untried') pins one by one in the selected row 'of 'net matrix' and calculates the nearest already connected pin, whose numbers are stored in the 'pin number array'. The distance between the two and numbers of the untried pin and already tried pin are stored in its local array 'B'. Initially the first pin of the net is considered as 'tried' or already connected. The local array B will consist of as many elements as there are unconnected or untried pins in the net. This number is returned in the parameter 'KJ'. The elements of 'B' are then sorted and that untried pin which corresponds to the element giving the least distance is treated as next 'start cell'. After invoking the routing procedures this pin is added to the 'pin number array'.

6.5 'MIC' FILE INSTRUCTIONS FOR PROGRAM EXECUTION

The reasons for MIC file instructions and Appendix VII were mentioned in Sec. 5.4. The 'Route1' instructions for example can be used to execute the router program for circuit after the execution of placement program or when a placement program output exists in the name of 'PLA1.OUT'. If independent input is given, the MIC file instruction can be still used by giving it the name 'PLA1 (or PLA2 etc).out'.

The router program may be followed by 'plot1' for example and then 'name1'. Otherwise 'name1' instructions must be executed by the simple command 'Do Name1'.

In the case of plotting routine again renaming of files and window and view port dimensions change in program data are required. If the necessary input files exist this can be carried out independently and then 'Name1' can be done. Otherwise this must follow a router program.

See Appendix VII for an example of relevant 'MIC' file instructions.

CHAPTER 7

CONCLUSION

All authors [1,2] agree that there must be a great deal more experimental and theoretical investigations of the layout techniques.

7.1 PLACEMENT

The relative value of each of the method seen depends upon the particular application and the amount of computer time/storage available. In addition time factor limits the variety of test cases that can be employed.

One good combination would be a fast initial placement method seen, coupled with an iterative placement improvement method. On the other hand the heuristic technique of applying vector forces for placement has the advantages:

- (a) Tends to be independent of initial data.
- (b) Connectability requirements are met along with placement.
- (c) The modelling of circuit board is very flexible and variety of wiring rules and requirements can be satisfied.
- (d) Computational efficiency is satisfactory
- (e) Provides a practical approach and in the cases tested does reduce line length considerably.

7.2 ROUTING

A number of techniques, both formal and heuristic have been developed. Their applicability and usefulness vary considerably with the problem under consideration.

A rough measure of the difficulty of the problem is as follows: Suppose 'L' is the total ('Manhattan') distance warranted by the interconnections once the wire list is determined, and let 'S' be the total number of grid segments (cells) available. Then, the ratio S/L is a measure of difficulty of the problem.

Like the implementation shown in Chapter 6, most are based on Lee's Algorithm. The variety stems from speedup techniques, net ordering and ordering a pins in the net.

Other techniques that have been tried include a graph-to-layout correspondence [9] in which a graph structure defines the connections and the router is capable of decoding this. Another method [8] assigns all horizontal-wires to one side and vertical wires on the other side, in the 'channel' between modules; segments of them are then allotted to nets by the router. This requires much less computer time and storage but limited in applications.

It can be said that no routing program can guarantee to layout all interconnections on board (since the ratio,

S/L may be too small). Hence the possibility of 'cant connects' must be allowed for, which must be kept at lowest level.

7.3 SUGGESTIONS FOR FUTURE IMPLEMENTATION

Placement:

- (a) More experimentation to gain use of some statistical studies in the present implementation.
- (b) Implementation of a monitor that would dynamically adjust program parameters such as stopping condition etc. depending on circuit.
- (c) Implementation of 'Expand' process so that placement can be made without regard to predetermined sites,
- (d) Use of Pythagorean vectors in sort-delete technique of calculating resultant instead of X and Y resultant separately as in the present implementation.
- (e) Calculation of mobility factor for each module instead of single mobility factor for all modules as in the present case.

Routing:

- (a) Speed up techniques for Lee's algorithm.
- (b) Modification to router (and corresponding modification to placement to output wire list i.e., cell pairs) to implement all wiring rules considered in placement.
- (c) Channel assignment method [8].
- (d) Addition of a software interactive facility to the plotting routine for Routing the unmade connections (after execution). This can use the cursor movement facilities presently available (for the Tektronix graphic terminal) with a kind of editor added to it.

BIBLIOGRAPHY AND REFERENCES

1. 'Automated placement of Multi-Terminal Components':
F. TAYLOR SCANLON - Proceedings of the SHARE-ACMO.
IEEE Design Automation work , June 1971
2. 'Design Automation of Digital Systems': Ed. by: MELVIN
A. BREUER, Prentice-Hall Inc., 1972.
3. 'Computer Oriented Operations Research': CHIHAAS - Tata
Mc-Graw Hill.
4. 'The Placement of Computer Logic Modules': J.S. NAMELAK -
J. SIAM, Vol. 13 (Oct. 1966), pp. 615-629.
5. 'Algorithms for Assignment and Transportation Problems':
J. MUNNICKRES - J. SIAM; vol. 5 (1957), pp. 32-38.
6. 'An Algorithm for Path Connections and its applications':
C.Y. LEE - IRE Transactions on Electronic Computers;
vol. EC-10, No. 3 (Sept. 1961); pp. 346-365.
7. 'Experiments with Wire-Routing on PCBs' : M. SERVIT and
J. SCHMIDT - CAD vol. 12, No. 5, Sept. 1980, pp. 213-234.
8. 'Wire Routing by Optimizing Channel Assignment within
Large Apertures': A. HASHIMOTO and J. STEVENS -
Proceedings of the SHARE. ACM. IEE Design Automation
Conference, June, 1971.
9. 'A Graph-to-Layout Correspondence in Single Layer
Routing': A.A. SZEPIENIEC - CAD vol. 13, No. 2,
March 1981, pp. 73-80.
10. Router Program in B.Tech. Thesis (E.E. 1981) on
Layout by RAJIV CHAUDHARY AND SANJIV TAYAL.

12
2 3 6 17 15 14 1 0
2 3 6 43 15 14 1 0

2 3 6 43 62 14 1 0
2 3 6 17 62 14 1 0
2 3 6 43 38 14 1 0
2 3 6 17 38 14 1 0
5 5 1 3 4 36 0 0 0
5 5 1 30 56 36 0 0 0
1 1 4 15 76 3 0 0 0
1 1 4 25 76 3 0 0 0
1 1 4 35 76 4 0 0 0
1 1 4 46 76 3 0 0 0

3 3
8 1 7 0 0 9 1 0 1 14 0 2 12 0 3 14 0 4 14 0 5 14 0 6 14 0
8 1 8 0 0 1 7 0 2 5 0 3 7 0 4 7 0 5 7 0 5 7 0 12 25 0
2 3 6 11 18 0 6 1 0
2 3 6 9 3 0 3 1 0
2 3 6 6 8 0 2 6 0
2 3 6 9 6 0 1 4 0
2 3 6 10 10 0 1 12 0
2 3 6 10 8 0 1 13 0
2 3 6 11 11 0 4 12 0
2 3 6 11 11 0 5 9 0
2 3 6 5 8 0 2 7 0
2 3 6 11 14 0 1 9 0
2 3 6 11 12 0 1 10 0
2 3 6 9 4 0 1 5 0
2 3 6 11 6 0 3 2 0
2 3 6 15 0 6 3 0
2 3 6 6 4 0 5 3 0
2 3 6 6 0 0 11 20 0
2 3 6 8 0 0 5 5 0
2 3 6 9 0 0 3 13 0
2 3 6 11 0 0 5 22 0
2 3 6 12 0 0 12 22 0
2 3 6 13 0 0 3 12 0
2 3 6 14 0 0 6 5 0
2 3 6 6 0 0 5 1 0
2 3 6 11 0 0 1 1 0
2 3 6 3 0 0 5 13 0
2 3 6 10 0 0 1 2 0
2 3 6 16 0 0 3 7 0
2 3 6 12 0 0 12 14 0
2 3 6 3 0 0 6 9 0
2 3 6 8 0 0 4 13 0
2 6 11 1
12 73

12 73
15 73
15 73
18 73
18 73
21 73
21 73
24 73
24 73
27 73
27 73
30 73
30 73
33 73
33 73
36 73
36 73
39 73
39 73
42 73
42 73
45 73
45 73
48 73
48 73

APPENDIX II (36 Pages)

```

00010 const
00020   MODULEMAX = 50;
00030   NETMAX = 50;
00040   TEMPMAX = 500;
00050   STABLE = 2;
00060   XSPAN = 60;
00070   YSPAN = 80;
00080   type
00090     DAAGTYP = (UNMARKED,CROSSED,ASSIGNED);
00100     ELEMENT = record
00110       DIST : integer;
00120       NISHAN : DAAGTYP
00130     end;
00140     STATYP = record
00150       TICKED,COVERED,ASSZEROD : boolean
00160     end;
00170     MODINDEX = 1..MODULEMAX;
00180     NETINDEX = 1..NETMAX;
00190     TEMPINDEX = 1..TEMPMAX;
00200     RULETYPE = (FIXED,FREE,XFIXED,YFIXED,XMAP,YMAP);
00210     CARTESIAN =
00220       record
00230         X,Y : integer
00240       end;
00250     COMPONENT =
00260       record
00270         KIND : MODTYPE;
00280         SIZE : CARTESIAN;
00290         POSITION : CARTESIAN;
00300         TEMPPTR : 0..TEMPMAX;
00310         FORCE : CARTESIAN;
00320         PINS : integer;
00330         ORIENT : 0..2;
00340         BARSUBS : integer
00350       end;
00360     BARCOMP =
00370       record
00380         SIZE : CARTESIAN;
00390         POSITION : CARTESIAN;
00400         PINS : integer;
00410         ORIENT : 0..1
00420       end;
00430     NETLIST =
00440       record
00450         MODULENO : MODINDEX;
00460         PINNO : integer;
00470         BARSUBNO : integer;
00480         NEXT : ^NETLIST
00490       end;

```

```

00500 NETCELL =
00510 record
00520   SIZE : MODINDEX ;
00530   RULE : RULETYPE ;
00540   PTR : ^NETLIST
00550 end ;
00560 TEMPCELL =
00570 record
00580   MODNO : MODINDEX ;
00590   CONNECTIONS : NETINDEX ;
00600   NEXT : 0..TEMPMAX
00610 end ;
00620
00630 var
00640   I,J,DIFFY,BARIND:integer;
00650   MODULE : array [MODINDEX] of COMPONENT ;
00660   NET : array [NETINDEX] of MPCELL ;
00670   TEMP : array [TEMPINDEX] of TEMPCELL ;
00680   NOUFMODULES : MODINDEX ;
00690   NOUFNETS : NETINDEX ;
00700   AVSIZE : CARTESIAN ;
00710   PROGRESS,MAXPDMET : MODINDEX ;
00720   TOTALLENGTH,LENGTH,AVLENGTH,AVVEJO,FREQ : integer ;
00730   PERCENT,ITRN : integer ;
00740   INC,STOP : boolean ;
00750   NOUFTERMS : integer ;
00760   TERMGCS : array [1..50] of CARTESIAN ;
00770   BARMAP : array [1..20] of MARCDP ;
00780   TERMLTYP : 0..1 ;
00790   TERMLER : integer ;
00800   MATSIZE : MODINDEX ;
00810   MATSITES : array [MODINDEX] of CARTESIAN ;
00820   MATRIX : array [MODINDEX,MODINDEX] of REAL ;
00830   STATROW,STATCOL : array [MODINDEX] of STATVP ;
00840   CELL : record
00850     X : integer ;
00860     Y : integer ;
00870     NO : integer
00880   end ;
00890
00900   V,W : array [1..4] of real ;
00910
00920   initprocedure ;
00930   begin
00940     W[1] := 0.0 ; V[1] := 0.0 ;
00950     W[2] := 60.0 ; V[2] := 0.75 ;
00960     W[3] := 0.0 ; V[3] := 0.0 ;
00970     W[4] := 80.0 ; V[4] := 1.0 ;
00980   end {initprocedure} ;

```

```

00990 procedure READIN;
01000   var
01010     I,K : MODINDEX;
01020     J : NETINDEX;
01030     T : 1..6;
01040     R : 1..3;
01050     P : METLIST;
01060   procedure READMODULES;
01070   begin
01080     BAKIND := 1;
01090     READLN(MODFMODULES);
01100     for I := 1 to MODFMODULES do
01110       with MODULF(I) do
01120         begin
01130           TEMPPTTR := 0;
01140           READ(T);
01150           case 1 of
01160             1 : KIND := FIXED;
01170             2 : KIND := XFIXED;
01180             3 : KIND := YFIXED;
01190             4 : KIND := XBAR;
01200             5 : KIND := YBAR;
01210             6 : KIND := YBAR;
01220           end;
01230           READ(SIZE,X,SIZE,Y,POSITION,X,POSITION,Y,P,AS,ORIENT,BARSUBS);
01240           for K := 1 to BARSUBS do
01250             begin
01260               READLN;
01270               with BARMAP(BAKIND) do
01280                 begin
01290                   READ(SIZE,X,SIZE,Y,POSITION,X,POSITION,Y,P,AS,ORIENT);
01300                   BAKIND := BAKIND+1;
01310                 end
01320               end;
01330             end;
01340             READLN;
01350           end;
01360         end(readModules);
01370       procedure FIXAVSIZE;
01380       begin
01390         AVSIZE.X:=0; AVSIZE.Y:=0;
01400         for K:=1 to MODFMODULES do
01410           begin
01420             if (MODULE(I).KIND=YBAR) then AVSIZE.X:=AVSIZE.X
01430             else AVSIZE.X:=AVSIZE.X+(MODULE(I).SIZE.X);
01440             if (MODULE(I).KIND=XBAR) then AVSIZE.Y:=AVSIZE.Y
01450             else AVSIZE.Y:=AVSIZE.Y+(MODULE(I).SIZE.Y);
01460           end;
01470         AVSIZE.X:=AVSIZE.X div MODFMODULES;
01480         AVSIZE.Y:=AVSIZE.Y div MODFMODULES;

```

```

01480 end{fixavsite};
01490 procedure FIXMATMODANDSITE;
01500 begin MATSIZE := 0;
01510 for I := 1 to NOOFMODULES do
01520 with MODULEFIJ do
01530 case KIND of
01540 FREE, XFIXED, YFIXED :
01550 begin
01560 MATSIZE := MATSIZE+1;
01570 MATMODS[MATSIZE] := I;
01580 MATSITES[MATSIZE].X := POSITION.X;
01590 MATSITES[MATSIZE].Y := POSITION.Y;
01600 end;
01610 end;
01620 FIXED, XHAK, YBAK :
01630 end
01640 end{FIXMATMODANDSITE};
01650 procedure READNETS;
01660 begin
01670 READLN(NOOFNETS);
01680 for J := 1 to NOOFNETS do
01690 with NETFIJ do
01700 begin
01710 PTR := 0;
01720 READ(SIZE);
01730 READ(R);
01740 case 1 : RULE := STAR;
01750 2 : RULE := SERIAL;
01760 3 : RULE := SPAN
01770 end;
01780 for K := 1 to SIZE do
01790 begin
01800 NEW(P);
01810 READ(P^.MODULEEND, P^.PIENO, P^.BANSURAO);
01820 P^.NEXT := PTR;
01830 PTR := P
01840 end;
01850 READLN
01860 end;
01870 end;
01880 { readnets }
01890 procedure FIXMAXPIN;
01900 begin
01910 MAXPINNET := 0;
01920 for J := 1 to NOOFNETS do with NETFIJ do
01930 if SIZE > MAXPINNET then MAXPINNET := SIZE
01940 end{fix maxpin};
01950 procedure READTERMLS;
01960 begin
01970 READLN (NOOFTERMLS, TERMLEN, TERMLTYP);

```



```

01970 for I := 1 to NOOFTERMS do
01980   READLN (TERMLCS[I].X,TERMLCS[I].Y)
01990 end;
02000 begin {readin}
02010   READMODULES;
02020   FIXAVSIZE;
02030   FIXATMODANDsite;
02040   READNETS;
02050   FIXMAXPIN;
02060   READTERMS;
02070 end;
02080 procedure DRAWRECT(P,SIZE:CARTESIAN; N:integer);
02090 var
02100   TEMP,I,X,Y:integer;
02110   STG: packed array[1..41] of char;
02120   begin
02130     X:=P.X-SIZE.X;Y:=P.Y-SIZE.Y;
02140     LINI(X,Y,0); X:=X+2*SIZE.X;LINI(X,Y);
02150     Y:=Y+2*SIZE.Y;LINI(X,Y);X:=X-2*SIZE.X;LINI(X,Y);
02160     Y:=Y-2*SIZE.Y;LINI(X,Y);
02170   end;
02180   SOFCTL(1); STG:= ' *';
02190   for I:=2 downto 1 do
02200     begin
02210       TEMP:=N mod 10; STGI[I]:=CHR(ORD('0')+TEMP);
02220       N:=N div 10;
02230     end;
02240   STRING(STG)
02250 end;
02260 {DRAWRECT}
02270 procedure CONVERGE;
02280 function DISTANCE(M1,M2:MODINDEX) : integer ;
02290 var
02300   X1,X2,Y1,Y2 : integer ;
02310   begin
02320     with MODULE[M1].POSITION do
02330       begin
02340         X1 := X ;
02350         Y1 := Y ;
02360       end ;
02370     with MODULE[M2].POSITION do
02380       begin
02390         X2 := X ;
02400         Y2 := Y ;
02410       end ;
02420     if (MODULE[M1].KIND=XBAR) or (MODULE[M2].KIND=XBAR)
02430     then
02440       begin
02450         if (MODULE[M1].KIND=YBAR) or (MODULE[M2].KIND=YBAR)

```

```

02950     else UPTO[K]:=K+1
02960     end;
02970     BARFLAG:=true;
02980     end;
02990     if not BARFLAG then
03000     begin
03010         SX := 0 ;
03020         SY := 0 ;
03030         for H := 1 to N do
03040             begin
03050                 SX := SX+MODULE[MAP[H]].POSITION.X ;
03060                 SY := SY+MODULE[MAP[H]].POSITION.Y ;
03070             end;
03080             CENTRE.X := SX/N ;
03090             CENTRE.Y := SY/N ;
03100             K := 1 ;
03110             MIN := K ;
03120             DMIN := ABS(CENTRE.X-MODULE[MAP[K]].POSITION.X)+
03130                 ABS(CENTRE.Y-MODULE[MAP[K]].POSITION.Y) ;
03140             while K < N do
03150                 begin
03160                     K := K+1 ;
03170                     D := ABS(CENTRE.X-MODULE[MAP[K]].POSITION.X)+
03180                         ABS(CENTRE.Y-MODULE[MAP[K]].POSITION.Y) ;
03190                     if D < DMIN
03200                     then
03210                         begin
03220                             MIN := K ;
03230                             DMIN := D
03240                         end
03250                     end;
03260                 for H := 1 to N-1 do
03270                     begin
03280                         FROM[H] := MIN ;
03290                         if H < MIN
03300                         then
03310                             UPTO[H] := H
03320                         else
03330                             UPTO[H] := H+1
03340                         end
03350                     end;
03360                 end;
03370             end { starreorder } ;
03380             procedure SERIALREOrder ;
03390             var
03400                 H,K,MIN,T : MODINDEX ;
03410                 D : integer ;
03420                 LENGTH : array [MODINDEX] of integer ;
03430             begin

```


03930
03940
03950
03960
03970
03980
03990
04000
04010
04020
04030
04040
04050
04060
04070
04080
04090
04100
04110
04120
04130
04140
04150
04160
04170
04180
04190
04200
04210
04220
04230
04240
04250
04260
04270
04280
04290
04300
04310
04320
04330
04340
04350
04360
04370
04380
04390
04400
04410

```

    UPTO[H] := H ;
    LENGTH[H] := DISTANCE(MAP[M],MAP[H])
  end ;
  for K := 1 to N-1 do
    begin
      MIN := K ;
      H := MIN ;
      while H < N-1 do
        begin
          H := H+1 ;
          if LENGTH[H] < LENGTH[MIN]
          then
            MIN := H
          end ;
          if K <> MIN
          then
            begin
              T := FROM[K] ;
              FROM[K] := FROM[MIN] ;
              FROM[MIN] := T ;
              T := UPTO[K] ;
              UPTO[K] := UPTO[MIN] ;
              UPTO[MIN] := T ;
              T := LENGTH[K] ;
              LENGTH[K] := LENGTH[MIN] ;
              LENGTH[MIN] := T
            end ;
            H := K ;
            while H < N-1 do
              begin
                H := H+1 ;
                D := DISTANCE(MAP[UPTO[K]],MAP[UPTO[H]]) ;
                if D < LENGTH[H]
                then
                  begin
                    LENGTH[H] := D ;
                    FROM[H] := UPTO[K]
                  end
                end
              end
            end
          end
        end
      end
    end
  end
  spanreorder ;
  procedure INSERT(H,K:MODINDEX) ;
  var
    PT : 0..TEMPMAX ;
    B : boolean ;
  begin
    PT := MODULE[H].TEMPPTR ;
    B := true ;
    while B do

```

```

04420 if PT = 0
04430 then
04440 begin
04450 with TEMPI[INDEX] do
04460 begin
04470 MODNO := K ;
04480 CONNECTIONS := 1 ; TEMPTR ;
04490 NEXT := MODULE[H].TEMPTR ;
04500 MODULE[H].TEMPTR := INDEX
04510 end ;
04520 INDEX := INDEX+1 ;
04530 A := false
04540 end
04550 else
04560 if TEMPI[PT].MODNO = K
04570 then
04580 begin
04590 TEMPI[PT].CONNECTIONS := TEMPI[PT].CONNECTIONS+1 ;
04600 B := false
04610 end
04620 else
04630 PT := TEMPI[PT].NEXT
04640 end
04650 { insert } ;
04660 begin
04670 INDEX := 1 ;
04680 for J:=1 to NODFMODULES do with MODULE[J] do TEMPTR:=0 ;
04690 for I:=1 to NODFNETS do
04700 begin
04710 N := NET[I].SIZE ;
04720 P := NET[I].PTR ;
04730 if ((PROGRESS=1) or (N<=PROGRESS)) then
04740 begin
04750 for J := 1 to N do
04760 begin
04770 MAP[J] := P*.MODULENO ;
04780 P := P*.NEXT
04790 end ;
04800 if (PROGRESS=1)
04810 then R := STAR
04820 else R := NET[I].RULE ;
04830 case R of
04840 STAR : STARREORDER ;
04850 SERIAL : SERIALREORDER ;
04860 SPAN : SPANREORDER
04870 end ;
04880 for J := 1 to N-1 do
04890 begin
04900

```

```

04910      11
04920      INSERT(MAP[FROM[J]],MAP[DEPTO[J]]);
04930      INSERT(MAP[UPTO[J]],MAP[FROM[J]])
04940      end
04950      end;
04960      LENGTH(TOTALLENGTH);
04970      MOVEND := 0; FREQ := 0; LENGTH := 0; STOP := false;
04980      AVLENGTH := 0; PERCENT := 0
04990      end { netreorder };
05000      procedure MOVE;
05010      var
05020          K : MODINDEX;
05030          AVX,AVY,MX,MY,MORFACTX,MORFACTY,FX,FY : real;
05040      procedure FORCE;
05050      type
05060          FORCEARRAY = array [1..500] of integer;
05070      var
05080          I,J : MODINDEX;
05090          P : 0..TEMPMAX;
05100          MG : MODINDEX;
05110          CW : NETINDEX;
05120          FX,FY : FORCEARRAY;
05130          BIT : array [MODINDEX] of boolean;
05140          FINT,FYINT : integer;
05150          function RESULTANT(var F:FORCEARRAY;IND : integer) : integer;
05160          var
05170              H : MODINDEX;
05180              LP,RP : MODINDEX;
05190              T : integer;
05200              B : boolean;
05210          begin
05220              repeat
05230                  B := true;
05240                  for H := 1 to IND-1 do
05250                      if F[H] > F[H+1]
05260                      then
05270                          begin
05280                              F[H] := F[H+1];
05290                              F[H+1] := F[H];
05300                              B := true;
05310                          end
05320                      until B;
05330                      LP := 1;
05340                      RP := IND;
05350                      while (F[LP] < 0) and (F[RP] > 0) do
05360                          begin
05370                              LP := LP+1;
05380                              RP := RP-1;

```

```

05400      end ?
05410      T := 0 ;
05420      for H := LP to RP do
05430          T := T+F(H) ;
05440          RESULTANT := T ;
05450      end { resultant } ;
05460      function SIGN(X:integer) : integer ;
05470      begin
05480          if X > 0
05490          then
05500              SIGN := +1
05510          else
05520              if X = 0
05530              then
05540                  SIGN := 0
05550              else
05560                  SIGN := -1
05570              { sign } ;
05580      function XVECTOR(P,O:MODINDEX) : integer ;
05590      begin
05600          case MODULE[P].KIND of
05610              FIXED, XBAR, YBAR, XFIXED
05620              : XVECTOR:=0 ;
05630              YFIXED, FREE
05640              : if (MODULE[P].KIND=FREE) or (MODULE[P].KIND=YFIXED)
05650                  then XVECTOR:=MODULE[P].POSITION.X - MODULE[P].POSITION.X
05660                  else
05670                      if (MODULE[P].KIND=YBAR) then XVECTOR:=0
05680                      else XVECTOR:=2*(MODULE[P].POSITION.Y-MODULE[P].POSITION.X)
05690                  end {XVECTOR} ;
05700      function YVECTOR(P,O:MODINDEX) : integer ;
05710      begin
05720          case MODULE[P].KIND of
05730              FIXED, XBAR, YBAR, YFIXED
05740              : YVECTOR:=0 ;
05750              FREE, XFIXED
05760              : if (MODULE[P].KIND=FREE) or (MODULE[P].KIND=XFIXED)
05770                  then YVECTOR:=MODULE[P].POSITION.Y - MODULE[P].POSITION.Y
05780                  else
05790                      if (MODULE[P].KIND=XBAR) then YVECTOR:=0
05800                      else YVECTOR:=2*(MODULE[P].POSITION.Y-MODULE[P].POSITION.Y)
05810                  end {YVECTOR} ;
05820      function XREPULSE(P,O:MODINDEX) : integer ;
05830      begin
05840          case MODULE[P].KIND of
05850              FIXED, XBAR, YBAR, XFIXED

```

```

05890      : XREPULSE:=0; FREE,YFIXED :
05900      if(MODULE(I).KIND=YBAR) then XREPULSE:=0
05910      else
05920      if
05930      ABS(MODULE(I).POSITION.X-MODULE(I).POSITION.Y) < 2*(AVSIZE.X)
05940      then
05950      XREPULSE := 2*(AVSIZE.X)
05960      else XREPULSE := 0
05970      end
05980      end(XREPULSE);
05990      {ALTERNATIVELY XREPULSE:=2*AVSIZE.X,TAKING PROPER SIGN ONLY}
06000      function YREPULSE(P,Q:MODINDEX):integer;
06010      begin
06020      case MODULE(I).KIND of
06030      FIXED,XBAR,YBAR,YFIXED
06040      :YREPULSE:=0; FREE,XFIXED :
06050      if(MODULE(I).KIND=YBAR) then YREPULSE:=0
06060      else
06070      if ABS (MODULE(I).POSITION.Y-MODULE(I).POSITION.Y)
06080      < 2*(AVSIZE.Y) then
06090      YREPULSE := 2*(AVSIZE.Y)
06100      else YREPULSE := 0
06110      end
06120      end(YREPULSE);
06130      {ALTERNATIVELY USE AVSIZE WITH SIGN}
06140      begin
06150      for I := 1 to NOOFMODULES do
06160      begin FXIND := 1; FYIND := 1;
06170      for J := 1 to NOOFMODULES do
06180      BIT(I,J) := false;
06190      P := MODULE(I).TEMPPIR ;
06200      while P <> 0 do
06210      begin
06220      MN := TEMP(I).MODNO ;
06230      CN := TEMP(I).CONNECTIONS ;
06240      for J := 1 to CN do
06250      begin FX(FXIND) := XVECTOR(I,MN);
06260      FYIND := FXIND+1
06270      end;
06280      for J := 1 to CN do
06290      begin FY(FYIND) := YVECTOR(I,MN);
06300      FYIND := FYIND+1
06310      end;
06320      BIT(MN) := true;
06330      P := TEMP(I).NEXT
06340      end;
06350      for J := 1 to NOOFMODULES do
06360      if not BIT(J) and (J <> I)
06370      then

```


14

```

06380      begin
06390          FX(FXIND) := XREPULSE(I,J);
06400          FXIND := FXIND + 1;
06410          FY(FYIND) := YREPULSE(I,J);
06420          FYIND := FYIND + 1;
06430      end
06440      FX(FXIND) := 0;
06450      FY(FYIND) := 0;
06460
06470      ***
06480      ***
06490      ***
06500      ***
06510      ***
06520      ***
06530      ***
06540      ***
06550      ***
06560      ***
06570      ***
06580      ***
06590      ***
06600      ***
06610      ***
06620      ***
06630      ***
06640      ***
06650      ***
06660      ***
06670      ***
06680      ***
06690      ***
06700      ***
06710      ***
06720      ***
06730      ***
06740      ***
06750      ***
06760      ***
06770      ***
06780      ***
06790      ***
06800      ***
06810      ***
06820      ***
06830      ***
06840      ***
06850      ***
06860      ***
06870      ***
06880      ***
06890      ***
06900      ***
06910      ***
06920      ***
06930      ***
06940      ***
06950      ***
06960      ***
06970      ***
06980      ***
06990      ***
07000      ***
07010      ***
07020      ***
07030      ***
07040      ***
07050      ***
07060      ***
07070      ***
07080      ***
07090      ***
07100      ***
07110      ***
07120      ***
07130      ***
07140      ***
07150      ***
07160      ***
07170      ***
07180      ***
07190      ***
07200      ***
07210      ***
07220      ***
07230      ***
07240      ***
07250      ***
07260      ***
07270      ***
07280      ***
07290      ***
07300      ***
07310      ***
07320      ***
07330      ***
07340      ***
07350      ***
07360      ***
07370      ***
07380      ***
07390      ***
07400      ***
07410      ***
07420      ***
07430      ***
07440      ***
07450      ***
07460      ***
07470      ***
07480      ***
07490      ***
07500      ***
07510      ***
07520      ***
07530      ***
07540      ***
07550      ***
07560      ***
07570      ***
07580      ***
07590      ***
07600      ***
07610      ***
07620      ***
07630      ***
07640      ***
07650      ***
07660      ***
07670      ***
07680      ***
07690      ***
07700      ***
07710      ***
07720      ***
07730      ***
07740      ***
07750      ***
07760      ***
07770      ***
07780      ***
07790      ***
07800      ***
07810      ***
07820      ***
07830      ***
07840      ***
07850      ***
07860      ***
07870      ***
07880      ***
07890      ***
07900      ***
07910      ***
07920      ***
07930      ***
07940      ***
07950      ***
07960      ***
07970      ***
07980      ***
07990      ***
08000      ***
08010      ***
08020      ***
08030      ***
08040      ***
08050      ***
08060      ***
08070      ***
08080      ***
08090      ***
08100      ***
08110      ***
08120      ***
08130      ***
08140      ***
08150      ***
08160      ***
08170      ***
08180      ***
08190      ***
08200      ***
08210      ***
08220      ***
08230      ***
08240      ***
08250      ***
08260      ***
08270      ***
08280      ***
08290      ***
08300      ***
08310      ***
08320      ***
08330      ***
08340      ***
08350      ***
08360      ***
08370      ***
08380      ***
08390      ***
08400      ***
08410      ***
08420      ***
08430      ***
08440      ***
08450      ***
08460      ***
08470      ***
08480      ***
08490      ***
08500      ***
08510      ***
08520      ***
08530      ***
08540      ***
08550      ***
08560      ***
08570      ***
08580      ***
08590      ***
08600      ***
08610      ***
08620      ***
08630      ***
08640      ***
08650      ***
08660      ***
08670      ***
08680      ***
08690      ***
08700      ***
08710      ***
08720      ***
08730      ***
08740      ***
08750      ***
08760      ***
08770      ***
08780      ***
08790      ***
08800      ***
08810      ***
08820      ***
08830      ***
08840      ***
08850      ***
08860      ***
08870      ***
08880      ***
08890      ***
08900      ***
08910      ***
08920      ***
08930      ***
08940      ***
08950      ***
08960      ***
08970      ***
08980      ***
08990      ***
09000      ***
09010      ***
09020      ***
09030      ***
09040      ***
09050      ***
09060      ***
09070      ***
09080      ***
09090      ***
09100      ***
09110      ***
09120      ***
09130      ***
09140      ***
09150      ***
09160      ***
09170      ***
09180      ***
09190      ***
09200      ***
09210      ***
09220      ***
09230      ***
09240      ***
09250      ***
09260      ***
09270      ***
09280      ***
09290      ***
09300      ***
09310      ***
09320      ***
09330      ***
09340      ***
09350      ***
09360      ***
09370      ***
09380      ***
09390      ***
09400      ***
09410      ***
09420      ***
09430      ***
09440      ***
09450      ***
09460      ***
09470      ***
09480      ***
09490      ***
09500      ***
09510      ***
09520      ***
09530      ***
09540      ***
09550      ***
09560      ***
09570      ***
09580      ***
09590      ***
09600      ***
09610      ***
09620      ***
09630      ***
09640      ***
09650      ***
09660      ***
09670      ***
09680      ***
09690      ***
09700      ***
09710      ***
09720      ***
09730      ***
09740      ***
09750      ***
09760      ***
09770      ***
09780      ***
09790      ***
09800      ***
09810      ***
09820      ***
09830      ***
09840      ***
09850      ***
09860      ***
09870      ***
09880      ***
09890      ***
09900      ***
09910      ***
09920      ***
09930      ***
09940      ***
09950      ***
09960      ***
09970      ***
09980      ***
09990      ***
10000      ***

```

```

06870 MY := FY ;
06880 end ;
06890 AVX := AVX/NOOFMODULES ;
06900 AVY := AVY/NOOFMODULES ;
06910 MOBFACX := AVX/(8*MX) ;
06920 MOBFACY := AVY/(8*MY) ;
06930 for K := 1 to NOOFMODULES do
06940 begin
06950   with MODULE[K] do
06960   begin
06970     POSITION.X := POSITION.X+ROUND(FORCE.X*MOBFACX) ;
06980     POSITION.Y := POSITION.Y+ROUND(FORCE.Y*MOBFACY) ;
06990     WRITE(POSITION.X);
07000     WRITE(POSITION.Y);
07010   end
07020 end
07030 end { move } ;
07040 procedure COMPARE ;
07050 var
07060   NEXTLENGTH:integer;
07070 begin
07080   STOP := false;
07090   LINELNGTH:=NEXTLENGTH;
07100   LENGTH:=LENGTH+NEXTLENGTH;
07110   AVLENGTH := LENGTH div MOVEND;
07120   PERCENT := ((TOTALLENGTH-AVLENGTH)*100) div TOTALLENGTH;
07130   if (PERCENT<=STABLE) then STOP := true;
07140   if FREQ=0 then
07150   begin
07160     (PERCENT>12) then FREQ :=1;
07170     (PERCENT>10) and (PERCENT<12) then FREQ := 2;
07180     (PERCENT>8) and (PERCENT<10) then FREQ := 3;
07190     (PERCENT>6) and (PERCENT<8) then FREQ := 4;
07200     (PERCENT>4) and (PERCENT<6) then FREQ := 6;
07210     (PERCENT>STABLE) and (PERCENT<4) then FREQ :=10
07220   end
07230   WRITELN('STOP:',STOP,'FREQ:',FREQ);
07240   WRITELN;
07250 end {compare};
07260 begin {CONVERGE}
07270   PROGRESS:=1;
07280   WRITELN('PROGRESS:',PROGRESS);
07290   WRITELN;
07300   ITRN := 0;
07310   for I:=1 to NOOFMODULES do with MODULE[I] do
07320     DRAWRECT(POSITION,SIZE,I);
07330     NETREORDER ;
07340     MOVE ;

```

07360
07370
07380
07390
07400
07410
07420
07430
07440
07450
07460
07470
07480
07490
07500
07510
07520
07530
07540
07550
07560
07570
07580
07590
07600
07610
07620
07630
07640
07650
07660
07670
07680
07690
07700
07710
07720
07730
07740
07750
07760
07770
07780
07790
07800
07810
07820
07830
07840

```

for I:=1 to NOOFMODULES do with MODULE(I) do
  DRAWRECT(POSITION,SIZE,I);
begin
  INC:=false;
  for I:=1 to NOOFNETS do
    if NET(I).SIZE=PROGRESS then INC := true;
  begin
    NETREORDER;
    ITRN := 0;
    repeat
      MOVE;
      CLADEV (3);
      for I:=1 to NOOFMODULES do
        with MODULE (I) do
          DRAWRECT (POSITION,SIZE,I);
          MOVEND:=MOVEND+1;
          ITRN := ITRN+1;
          COMPARE;
          if not STOP and (MOVEND=FREE)
            then NETREORDER
              until (STOP or (ITRN=3))
            end
          end;
    end(CONVERG);
  procedure ASSIGNMENT;
  procedure FIXMATRIX;
  const
    MAXINT = 999;
  var
    I,J: integer;
  begin(FIXMATRIX);
    for I:=1 to MAISIZE do
      begin
        for J:=1 to MATSIZE do
          with MODULE(MATMDS(I)) do
            begin
              MATRIX(I,J).NISHAN := UNMARKED;
              if KIND=FREE then
                +MATRIX(I,J).DIST := ABS(POSITION.X-MATSITES(I,J).X)
                +ABS(POSITION.Y-MATSITES(I,J).Y);
              if KIND=FIXED then
                if POSITION.X=MATSITES(I,J).X then
                  MATRIX(I,J).DIST := ABS(POSITION.Y-MATSITES(I,J).Y)
                else

```

```

07850
07860
07870
07880
07890
07900
07910
07920
07930
07940
07950
07960
07970
07980
07990
08000
08010
08020
08030
08040
08050
08060
08070
08080
08090
08100
08110
08120
08130
08140
08150
08160
08170
08180
08190
08200
08210
08220
08230
08240
08250
08260
08270
08280
08290
08300
08310
08320
08330

17
MATRIX(I,J).DIST := MAXINT;
if KIND=FIXED then
  if POSITION.X=MAISITES(I,J).Y then
    MATRIX(I,J).DIST := ABS(POSITION.X-MAISITES(I,J).Y)
  else
    MATRIX(I,J).DIST := MAXINT
  {
    WRITE (MATRIX(I,J).DIST)
  }
end {
  WRITE (TTY); BREAK
}
end
end {FIXMATRIX};
procedure HUNGARIAN;
var
  SUCCESS : boolean; UNMRK : MODINDEX;
  procedure FIXSTATUS;
  var
    I,J : integer;
  begin
    FIXSTATUS;
    for I := 1 to MATSIZE do
      begin
        STATROW(I).TICKED := false;
        STATROW(I).COVERED := false;
        STATCOL(I).TICKED := false;
        STATCOL(I).COVERED := false;
        for J := 1 to MATSIZE do
          MATRIX(I,J).NISHAN := UNMRKED
        end
      end
    end {FIXSTATUS};
  procedure MATPRINT;
  var
    I,J : integer;
  begin
    MATPRINT;
    for I := 1 to MATSIZE do
      begin
        for J := 1 to MATSIZE do
          WRITE ( TTY,MATRIX(I,J).DIST)
        end;
        WRITELN (TTY); BREAK
      end;
    end {MATPRINT};
  procedure STATRPRINT;
  var
    I,J : integer;
  begin
    STATRPRINT;
    for I := 1 to MATSIZE do
      begin
        WRITELN (TTY);
        if TICKED then WRITELN ( TTY,'ROW',I,'TICKED');
        if COVERED then WRITELN ( TTY,'ROW',I,'COVERED');
      end
    end
  end
end

```

```

08340      18
08350      if ASSZEROD then WRITELN ( TTY, 'ROW', I, 'HAS ASSIGNMENT');
08360      WRITELN ( TTY); BREAK
08370      end
08380      end{STATRPRINT};
08390      procedure STATCPRINT;
08400      var
08410      I : integer;
08420      begin{STATCPRINT}
08430      for I := 1 to MATSIZE do with STATCOL(I) do
08440      begin
08450      WRITELN ( TTY);
08460      if TICKED then WRITELN ( TTY, 'COL', I, 'TICKED');
08470      if COVERED then WRITELN ( TTY, 'COL', I, 'COVERED');
08480      if ASSZEROD then WRITELN ( TTY, 'COL', I, 'HAS ASSIGNMENT');
08490      WRITELN (TTY); BREAK
08500      end
08510      end{STATCPRINT};
08520      procedure ZEROPRINT;
08530      var
08540      I, J : integer;
08550      begin{ZEROPRINT}
08560      for I := 1 to MATRIXII, JJ do
08570      if (DIST=0) then
08580      begin
08590      if (NISHAN=UNMARKED) then
08600      WRITELN ( TTY, 'UNMARKED ZERO IN ROW', I, 'COL', JJ);
08610      if (NISHAN=CROSSED) then
08620      WRITELN ( TTY, 'CROSSED ZERO IN ROW', I, 'COL', JJ);
08630      if (NISHAN=ASSIGNED) then
08640      WRITELN ( TTY, 'ASSIGNED ZERO IN ROW', I, 'COL', JJ)
08650      end;
08660      end;
08670      WRITELN ( TTY, 'SUCCESS', 'AND UNMARKEDZEROS:', 'UNMARK');
08680      WRITELN (TTY);
08690      end{ZEROPRINT};
08700      procedure ROWMIN;
08710      var
08720      I, J, MIN : integer;
08730      begin{ROWMIN}
08740      for I := 1 to MATSIZE do
08750      begin
08760      MIN := MATRIXII, I, DIST;
08770      for J := 1 to MATSIZE do
08780      if MATRIXII, JJ, DIST < MIN then MIN := MATRIXII, JJ, DIST;
08790      for J := 1 to MATSIZE do
08800      MATRIXII, JJ, DIST := MATRIXII, JJ, DIST-MIN
08810      end
08820      end{ROWMIN};
08830      end{COLMIN};

```

```

08830 var JMIN : integer;
08840 begin{COLMIN}
08850 for J := 1 to MATSIZE do
08860 begin
08870 MIN := MATRIX(I,J).DIST;
08880 for I := 1 to MATSIZE do
08890 if MATRIX(I,J).DIST < MIN then MIN := MATRIX(I,J).DIST;
08900 for I := 1 to MATSIZE do
08910 if MATRIX(I,J).DIST := MATRIX(I,J).DIST - MIN
08920 end
08930 end{COLMIN};
08940 procedure ASSROW;
08950 var ROWCOND : array[MODINDEX] of boolean;
08960 ROWSDONE : boolean;
08970 REMEMB : 0..MODULEMAX;
08980 I,J,K : integer;
08990 begin{ASSROW}
09000 ROWSDONE := false;
09010 for I := 1 to MATSIZE do
09020 if ROWCOND[I] := false;
09030 repeat
09040 for I := 1 to MATSIZE do
09050 begin
09060 UNMRK := 0; REMEMB := 0;
09070 for J := 1 to MATSIZE do
09080 if (MATRIX(I,J).DIST=0) and (MATRIX(I,J).NISHAN=UNMARKED)
09090 then
09100 begin UNMRK := UNMRK+1; REMEMB := J
09110 end;
09120 if UNMRK=1 then
09130 begin
09140 MATRIX(I,REMEMB).NISHAN := ASSIGNED;
09150 for K := 1 to MATSIZE do
09160 if (MATRIX(K,REMEMB).DIST=0) and (K < I)
09170 then MATRIX(K,REMEMB).NISHAN := CROSSED;
09180 STATROW[I].ASSZEROD := true;
09190 STATCOL[REMEMB].ASSZEROD := true
09200 end
09210 end;
09220 I := I+1;
09230 repeat (ROWCOND[I]=STATROW[I].ASSZEROD)
09240 if then
09250 begin
09260 I := I+1;
09270 ROWSDONE := true
09280 end
09290 end
09300 end
09310

```

```

093320      end
093330    else
093340      begin
093350        for J := 1 to MATSIZE do ROWCOND[J] := STATROW[J].ASSZEROD;
093360        ROWSDONE := false;
093370        I := MATSIZE+1
093380      end
093390      until I=MATSIZE+1
093400    until ROWSDONE
09410    end{ASSROW};
09420  procedure ASSCOL;
09430  var
09440    COLCOND : array[MODINDEX] of boolean; COLSDONE : boolean;
09450    I,K,J : integer; REMEMB : 0..MODULUMAX;
09460  begin
09470    COLSDONE := false;
09480    for J := 1 to MATSIZE do COLCOND[J] := false;
09490    repeat
09500      for J := 1 to MATSIZE do
09510        begin
09520          UNMRK := 0; REMEMB := 0;
09530          for I := 1 to MATSIZE do
09540            if (MATRIX[I,J].DIST=0) and (MATRIX[I,J].NISHAN=UNMARKED)
09550              then
09560                begin UNMRK := UNMRK+1; REMEMB := I
09570                end;
09580            if (UNMRK=1) then
09590              begin
09600                MATRIX[REMEMB,J].NISHAN := ASSIGNED;
09610                for K := 1 to MATSIZE do
09620                  if (MATRIX[REMEMB,K].DIST=0) and (K<>J)
09630                    then MATRIX[REMEMB,K].NISHAN := CROSSED;
09640                STATCOL[J].ASSZEROD := true;
09650                STATROW[REMEMB].ASSZEROD := true
09660              end
09670            end;
09680            J := 1;
09690          repeat
09700            if COLCOND[J]=STATCOL[J].ASSZEROD
09710              then
09720                begin
09730                  J := J+1;
09740                  COLSDONE := true
09750                end
09760              else
09770                begin
09780                  for I := 1 to MATSIZE do COLCOND[J] := STATCOL[J].ASSZEROD;
09790                  COLSDONE := false;
09800                  J := MATSIZE+1

```

```

end;
until (J=MATSIZE+1)
until COLSDONE
begin
  { ASSCOL }
  procedure CONDITIONCheck;
  var
    I, J, K : integer;
    ARBITROW, ARBITCOL : 0..MODULEMAX;
  begin
    I := 1; UNMRK := 0; ARBITROW := 0; ARBITCOL := 0;
    SUCCESS := false;
    repeat
      if STATROW[I].ASSZEROD
      then
        begin
          SUCCESS := true; I := I+1
        end
      else
        begin
          SUCCESS := false; I := MATSIZE+1
        end
      end
    until (I=MATSIZE+1);
    { { { {
    if SUCCESS then
      begin MATPRINT; ZEROPRINT;
      end; } } } }
    if (not SUCCESS)
    then
      begin
        I := 1; J := 1;
        repeat
          if (MATRIX[I, J].DIST=0) and (MATRIX[I, J].NISHAN=UNMARKED)
          then
            begin
              UNMRK := UNMRK+1;
              if (UNMRK>1) then
                begin
                  ARBITROW := I;
                  ARBITCOL := J;
                  J := MATSIZE+1;
                  I := MATSIZE+1
                end
              end
            end
          else J := J+1
          until J=MATSIZE+1; I := I+1
          until I >= MATSIZE+1;
          if (UNMRK>1)
          then
            begin
              MATRIX[ARBITROW, ARBITCOL].NISHAN := ASSIGNED;
              STATROW[ARBITROW].ASSZEROD := true;
            end
          end
        end
      end
    end
  end;
end;

```



```

10300 STATCOLIARBITCOLI.ASSZEROD := true;
10310 for K := 1 to MATSIZE do
10320 begin
10330 if (MATRIXIARBITROW,KI.DIST=0) and (K<>ARBITCOL)
10340 then MATRIXIARBITROW,KI.NISHAN := CROSSED;
10350 if (MATRIXIK,ARBITCOLI.DIST=0) and (K<>ARBITROW)
10360 then MATRIXIK,ARBITCOLI.NISHAN := CROSSED
10370 end
10380 end
10390 end {CONDITIONCHECK};
10400 procedure REDUCEMATRIX;
10410 procedure TICKLINES;
10420 var
10430 ROWTICK,COLTICK : array [MODINDEX] of boolean;
10440 FLAG,TICKSDONE : boolean; I,J : integer;
10450 TICKLINES;
10460 for I := 1 to MATSIZE do
10470 begin ROWTICK[I] := false; COLTICK[I] := false
10480 end;
10490 for I := 1 to MATSIZE do
10500 begin
10510 FLAG := true;
10520 for J := 1 to MATSIZE do
10530 if (MATRIXII,J).NISHAN<>ASSIGNED) and FLAG
10540 then
10550 FLAG := true
10560 else
10570 FLAG := false;
10580 if FLAG
10590 then STATROW[I].TICKED := true
10600 end;
10610 repeat
10620 J := 1 to MATSIZE do
10630 if (not STATCOL[J].TICKED)
10640 then
10650 begin FLAG := false;
10660 for I := 1 to MATSIZE do
10670 if (STATROW[I].TICKED) and
10680 (MATRIXII,J).DIST=0) then
10690 FLAG := true;
10700 if FLAG then
10710 STATCOL[J].TICKED := true
10720 end;
10730 for I := 1 to MATSIZE do
10740 if (not STATROW[I].TICKED)
10750 then
10760 begin
10770 FLAG := false; for J := 1 to MATSIZE do
10780

```

```

10790
10800
10810
10820
10830
10840
10850
10860
10870
10880
10890
10900
10910
10920
10930
10940
10950
10960
10970
10980
10990
11000
11010
11020
11030
11040
11050
11060
11070
11080
11090
11100
11110
11120
11130
11140
11150
11160
11170
11180
11190
11200
11210
11220
11230
11240
11250
11260
11270

23
      if (STATCOL(J).TICKED) and
      (MATRIX(I,J).NISHAN=ASSIGNED)
      then FLAG := true;
      if FLAG then STATROW(I).TICKED := true
      end;
      TICKSDONE := false; I := 1;
      repeat
      if (STATROW(I).TICKED=ROWTICK(I)) and (STATCOL(I).TICKED=COLP1
      then
      begin TICKSDONE := true; I := I+1
      end
      else
      begin TICKSDONE := false;
      for J := 1 to MATSIZE do
      begin
      ROWTICK(J) := STATROW(J).TICKED;
      COLTICK(J) := STATCOL(J).TICKED;
      end;
      I := MATSIZE+1
      end
      until (I=MATSIZE+1)
      until TICKSDONE
      end{TICKLINES};
      procedure COVERLINES;
      var COVLIN, I : integer; COVLIN := 0;
      begin{coverlines} COVLIN := 0;
      for I := 1 to MATSIZE do
      begin
      if (not STATROW(I).TICKED) then
      begin STATROW(I).COVERED := true; COVLIN := COVLIN+1
      end;
      if STATCOL(I).TICKED then
      begin STATCOL(I).COVERED := true; COVLIN := COVLIN+1
      end
      end;
      if (COVLIN>=MATSIZE)
      { then WRITELN ( IY, COVLIN, 'linescovered-error', J )
      { MATPRINT; ZEROPRINT; STATPRINT; STATCPRINT)
      end{coverlines};
      procedure DOFINALMAT;
      var MIN, I, J : integer;
      begin{do finalmata;
      MIN := 9999;
      for I := 1 to MATSIZE do
      begin
      if (not STATROW(I).COVERED)
      then for J := 1 to MATSIZE

```

```

280  if (not STATCOL(I).COVERED) and (MATRIX(I,J).DIST<MIN)
290  then MIN := MATRIX(I,J).DIST
300
310  end;
320  for I := 1 to MATSIZE do
330  if (not STATROW(I).COVERED) then
340  for J := 1 to MATSIZE do
350  MATRIX(I,J).DIST := MATRIX(1,J).DIST - MIN;
360  for J := 1 to MATSIZE do
370  if (STATCOL(J).COVERED) then
380  begin
390  for I := 1 to MATSIZE do
400  MATRIX(I,J).DIST := MATRIX(1,J).DIST+MIN
410  end;
420  FIXSTATUS;
430  end {dofinalmat};
440  begin {reduce matrix}
450  %ZEROPRINT;\
460  TICKLINES;
470  COVERLINES;
480  %STATCPRINT;\
490  %STATCPRINT;\
500  DOFINALMAT
510  end {reduce matrix};
520  procedure ASSIGN;
530  var I,J : integer;
540  begin {assign}
550  for I := 1 to MATSIZE do
560  if (STATROW(I).ASSZEROD) then
570  begin
580  for J := 1 to MATSIZE do
590  if (MATRIX(I,J).WISHAN=ASSIG.FD) then
600  begin
610  MODULE[MATMODS(I)].POSITION.X := MATSITES(I).X;
620  MODULE[MATMODS(I)].POSITION.Y := MATSITES(I).Y;
630  end
640  end
650  end {assign};
660  begin {HUNGARIAN}
670  ROWMIN;
680  COLMIN;
690  repeat
700  FIXSTATUS;
710  %MATPRNT;\
720  SUCCESS := false;
730  UNMRK := 2;
740  while (not SUCCESS) and (UNMRK>1) do
750  begin
760  ASSROW;

```

```

11770 ASSCOL;
11780 CONDITIONCheck
11790 end;
11800 if SUCCESS then ASSIGN
11810 else
11820   if (UNMRK=0) then REDUCEMATRIX
11830   else WRITELN ('error in 3 conditions');
11840   until HUNGARIAN;
11850   end HUNGARIAN;
11860   begin {ASSIGNMENT};
11870   FIXMATRIX;
11880   HUNGARIAN;
11890   end {ASSIGNMENT};
11900   procedure DRAWPLACEMENT;
11910   var
11920     I : integer;
11930     TERMLSDONE : boolean;
11940     procedure MARKPINS (P,SIZE:CARTESIAN;M,NPIN,ORNT:integer);
11950     const
11960       PINLEN = 0.5;
11970     var
11980       L,J,K : integer;
11990       SIG : packed array[1..4] of char;
12000       PLEN : real;
12010       STRT : record
12020         X,Y : real;
12030       end;
12040       begin MARKPINS\
12050         K := NPIN;
12060         DIFFY:=2;
12070         if ORNT<>0 then
12080           begin
12090             if ORNT=1 then
12100               begin
12110                 DIFFY := -DIFFY;PLEN := -PINLEN;
12120                 STRT.X := P.X-SIZE.X; STRT.Y := P.Y+SIZE.Y
12130               end
12140             else
12150               begin
12160                 PLEN := PINLEN;
12170                 STRT.X := P.X+SIZE.X;
12180                 STRT.Y := P.Y-SIZE.Y
12190               end;
12200             for J := 1 to 2 do
12210               begin
12220                 for I := 1 to K div 2 do
12230                   begin
12240                     LINE(STRT.X,STRT.Y,0);
12250                     LINE(STRT.X+PLEN,STRT.Y);
12260                     STRT.Y := STRT.Y+DIFFY

```

```

12260 end; y := STRT.Y-DIFFY;
12270 STRT.X := STRT.X+DIFFY; STRT.X := STRT.X+(SIZE.X*2)
12280 if ORNT=1 then STRT.X := STRT.X-(SIZE.X*2);
12290 else STRT.X := STRT.X+(SIZE.X*2);
12300 DIFFY := -DIFFY; PLEN := -PLEN;
12310 end; FOR JN;
12320 if DPNT=1 then
12330 STRT.X := P.X-(SIZE.X*2)-PLEN;
12340 else STRT.X := P.X+(SIZE.X*2)+PLEN;
12350 SIG := STRT.X;
12360 STG(1) := '1';
12370 LINE(STRT.X, STRT.Y, 0);
12380 SOFCTL(1);
12390 STRING(STG)
12400 end
12410 end; MAKKPIN;
12420 procedure MARKTERMLS;
12430 var
12440 STG : packed array[1..4] of char;
12450 I : integer;
12460 STRT : record
12470 X, Y : integer
12480 end;
12490
12500 begin
12510 I := 1 to NODFTERMLS do
12520 begin
12530 STRT.X := TERMLCS(1).X;
12540 STRT.Y := TERMLCS(1).Y;
12550 if TERMLTYP=1 then
12560 begin
12570 LIMI (STRT.X, STRT.Y, 0);
12580 LIMI (STRT.X, STRT.Y+TERALEN, 1)
12590 end
12600 else
12610 begin
12620 LIMI (STRT.X, STRT.Y, 0);
12630 LIMI (STRT.X+TERMLEN, STRT.Y, 1)
12640 end;
12650 if I=1 then
12660 begin
12670 STG(1) := '1';
12680 LIMI (STRT.X, STRT.Y, 0);
12690 if TERMLTYP=1 then LINE(STRT.X-0.5, STRT.Y, 0)
12700 else LINE(STRT.X, STRT.Y-0.5, 0);
12710 SOFCTL(1);
12720 STRING(STG)

```

```

12750 end(MARKTERMLS);
12760 begin
12770   TERMLSDONE := false;
12780   for I:= 1 to NOOFFMODULES do
12790     with MODULE(I) do
12800       if (ORIENT<>0) then
12810         begin
12820           DRAWRECT (POSITION,SIZE,I);
12830           MAKKPINS (POSITION,SIZE,I,PINS,ORIENT)
12840         end
12850       else
12860         if (PINS=0) then
12870           DRAWRECT (POSITION,SIZE,I)
12880         else
12890           if (not TERMLSDONE) then
12900             begin
12910               MARKTERMLS;
12920               TERMLSDONE := true
12930             end
12940           end
12950         end $DRAW PLACEMENT;
12960       procedure WRITECELL (M:MODINDEX; P:integer);
12970       var
12980         SGN, I : integer;
12990         DIFFY :=ABS (DIFFY);
13000         with MODULE(M) do
13010           if ORIENT<>0 then
13020             begin
13030               if ORIENT=1 then SGN := 1
13040               else SGN := -1;
13050               if P<=PINS div 2 then
13060                 begin
13070                   CELL.X := POSITION.X-SGN*SIZE.X;
13080                   CELL.Y := YSPAN - ( POSITION.Y - SGN*SIZE.Y+SGN*SIZE.Y-SGN*(P-1)*0.1FFY);
13090                   CELL.NO := (CELL.Y-1)*XSPAN+CELL.X
13100                 end
13110               else
13120                 begin
13130                   CELL.X := POSITION.X+(SGN*SIZE.X);
13140                   CELL.Y := YSPAN - ( POSITION.Y - SGN*SIZE.Y+SGN*SIZE.Y+(PINS div 2 +1))*DIFFY);
13150                   CELL.NO := (CELL.Y-1)*XSPAN+CELL.X
13160                 end;
13170               WRITELN (CELL.NO)
13180             end;
13190             if (MODULE(M).ORIENT=0 ) and ( MODULE(M).PINS<>0 ) then
13200               begin
13210                 if (P=0 ) then
13220                   for I:= 1 to NOOFFTERMLS do
13230

```

```

13240 CELL.X := TERMLDCS[I].X;
13250 CELL.Y := YSPAN - TERMLDCS[I].Y;
13260 CELL.NO := (CELL.Y-1)*XSPAN + CELL.X;
13270 WRITELN(CELL.NO)
13280 end
13290 else
13300 begin
13310 CELL.X := TERMLDCS[P].X;
13320 CELL.Y := YSPAN - TERMLDCS[P].Y;
13330 CELL.NO := (CELL.Y-1)*XSPAN + CELL.X;
13340 WRITELN(CELL.NO)
13350 end
13360 end
13370 end
13380 procedure WRITEBARCELL(M:MODINDEX; P,S:integer);
13390 var
13400 MAPIND,I,J,K ,SGN: integer;
13410 begin
13420 MAPIND:= S;
13430 K:=N;
13440 for I := (K-1) downto 1 do
13450 MAPIND := MAPIND + MODULCELL.BARSUBS;
13460 if P=0 then
13470 for J := (MAPIND+1) to (MAPIND+MODULCELL.BARSUBS) do
13480 with BARMAP[I] do
13490 begin
13500 if ORIENT=1 then SGN := 1
13510 else SGN:=-1;
13520 for J := 1 to PINS do
13530 begin
13540 if JK=PINS div 2 then
13550 begin
13560 CELL.X := POSITION.X-SGN*SIZE.X;
13570 CELL.Y := YSPAN - (POSITION.Y+SGN*SIZE.Y-SGN*(P-1)*DIFFY);
13580 CELL.NO := (CELL.Y-1)*XSPAN+CELL.X
13590 end
13600 else
13610 begin
13620 CELL.X := POSITION.X+(SGN*SIZE.X);
13630 CELL.Y := YSPAN - (POSITION.Y - SGN*SIZE.Y+(SGN*
13640 (P-(PINS div 2 + 1)))
13650 CELL.NO := (CELL.Y-1)*XSPAN+CELL.X
13660 end;
13670 WRITELN(CELL.NO)
13680 end
13690 end
13700 else
13710 with BARMAP[MAPIND] do
13720 begin

```

```

13730 if P<=PINS div 2 then
13740   begin
13750     CELL.X := POSITION, X-SGN*SIZE.X;
13760     CELL.Y := YSPAN - (POSITION.Y+SGN*SIZE.Y-SGN*(P-1)*OTFFY);
13770     CELL.NO := (CELL.Y-1)*XSPAN+CELL.X
13780   end
13790   else
13800     begin
13810       CELL.X := POSITION, X+(SGN*SIZE.X);
13820       CELL.Y := YSPAN - (POSITION.Y - SGN*SIZE.Y+(P-(PINS div 2 +1)))
13830       CELL.NO := (CELL.Y-1)*XSPAN+CELL.X
13840     end;
13850     WRITELN (CELL.NO)
13860   end
13870 end (BARCELLS);
13880 procedure WRITEALLPincells;
13890 var
13900   I,J : integer;
13910   TERMCELLDone : boolean;
13920   TERMCELLDone := false;
13930   for I:=1 to NOOFMODULES do
13940     for with MODULE(I) do
13950       begin
13960         WRITELN;WRITELN;
13970         if (ORIENT<>0) then
13980           for J:=1 to PINS do WRITECELL (I,J)
13990         else
14000           if ((PINS=0) and (BARSUBS<>0)) then
14010             WRITEBARCELL (I,0,0)
14020           else
14030             if (PINS<>0) then
14040               begin
14050                 if (not TERMCELLDone) then WRITECELL (I,0);
14060                 TERMCELLDone := true
14070               end
14080             end;
14090             WRITELN;WRITELN
14100             end @WRITEALLPincells;
14110           procedure WRITENETPincells;
14120             var
14130               I,J : integer;
14140               N : NETINDEX;
14150               P : NETLIST;
14160             begin
14170               for I := 1 to NOOFNETS do
14180                 begin
14190                   WRITELN; WRITELN;
14200                   N := NET(I).SIZE;
14210

```



```

14220 P := NET(I).PTR;
14230 for J := 1 to N do
14240   begin
14250     if P^.BARSUBNO=0 then
14260       WRITECELL (P^.MODULENO,P^.PINNO)
14270     else
14280       WRITEBARCELL (P^.MODULENO,P^.PINNO,P^.BARSUBNO);
14290     P := P^.NEXT
14300   end;
14310   WRITELN ('0')
14320 end
14330 end %WRITENETPINCELLSI\;
14340 begin %PLACEMENT\;
14350   NITDEV (3);
14360   WINDW (W);
14370   VBORT (V);
14380   READIN;
14390   CONVERGE;
14400   ASSIGNMENT;
14410   READLN (TTY,I) %TO CONFIRM USER UK REFUSE DRAWING\;
14420   CLRDEV (3);
14430   DRAWPLACEMENT;
14440   WRITELN (XSPAN);
14450   WRITELN (YSPAN);
14460   WRITEALLPINCELLS;
14470   WRITELN (99998);
14480   WRITENETPINCELLS;
14490   WRITELN (99999);
14500   READLN (TTY,J) %COFIRM USER OK PD CLEAR\;
14510   CLRDEV (3)
14520 end %PLACEMENT\.
```

60
80

700
820
940
1060
1180
1300
1420
1426
1306
1186
1066
946
826
706

3494
3614
3734
3854
3974
4094
4214
4220
4100
3980
3860
3740
3620
3500

3520
3640
3760
3880
4000
4120
4240
4246
4126
4006
3886
3766
3646

3526

674
794
914
1034
1154
1274
1394
1400
1280
1160
1040
920
800
680

2140
2260
2380
2500
2620
2740
2860
2866
2746
2626
2506
2386
2266
2146

2114
2234
2354
2474
2594
2714
2834
2840
2720
2600
2480
2360
2240
2120

372
372
375
378
378
381
381
384
384
387
387
390
390
393
393
395
396
399
399
402
402
405
405
408
408

99998

2120
2146
680
3526
3740
706

0 372

408
2834
2860
1394
4240
3974
1420

0 2114
396

0 3734
2234

0 3520
375

0 4094
2840

0 1060
378

0 946
384

0 826
381

0 920
1066

5

0 2746
1040

0 4214
2866

0 1306
390

0 1186
387

0 1180
375

0 3640
1300

0 2354
3380

0 2380
2474

0 399
2740

0 2620
4220

0 3646
4100

0 2260
3886

0 402
2386

0 3766
3620

0 2594
3500

0 2140
2714

0 700
3860

0 2266
940

0 820
3980

0 1240
3260

0 390
2360

0 3854
2720
3760

0 2506
800
1425
99999

APPENDIX IV (14 Pages)

```

00010 PROGRAM PCALAYOUT(INPUT,OUTPUT,OUT) ;
00020 type
00030 RTUPLE = record
00040     SELECT : 0..100 ;{NO. CORRESP TO SIGNALSET}
00050     CHAINCOORD : integer ;{CELL FROM WHICH}
00060     PIN : 0..2 ;{CELLSTATUS FOR OTHERSIDE}
00070 end ;
00080 SPAN = record
00090     DIST : integer ;{FOR ORDERING}
00100     XSPAN : integer ;
00110     YSPAN : integer ;
00120     ORDER : 1..100 ;{ACTUAL ORDER OF SIGNAL SET}
00130 end ;
00140 INOUY = record
00150     SELECT : 0..1 ;{SELECTED PIN FOR LAYOUT}
00160     DATA : integer ;{CELL NO TO BE CONNECTED}
00170 end ;
00180 AB = record
00190     DATA : integer ;
00200     SIGNAL : 1..100 ;{NO OF SIGNAL SETS}
00210     POSITION : 1..50 ;{POS. OF SIGNAL SET}
00220 end ;
00230 var
00240     LISTLPT1, LISTLPT2 : integer ;{PTRS TO LIST}
00250     SECT : integer ;{PTR TO ARRAY SECSIDE}
00260     NOCPATH : integer ;{NO CORRESP. TO SIGNAL SET}
00270     XCOORD, YCOORD : integer ;{NO OF CELLS IN XY DIRECTION}
00280     STARTC, ENDC : integer ;{TERMINAL CELLS}
00290     NSIGNAL : integer ;{TOTAL SIG SETS}
00300     T1, T2, T3, T4, J : integer ;
00310     ISIG, JPOS, AC, PREVIOUS, MIN, KJ, I, U, N : integer ;
00320     LISTL : array [0..14000] of integer ;{CELLS UNDER CONSIDERATION}
00330     CELLSTATUS : array [0..14000] of RIUPLE ;
00340     OUT : file of char ;
00350     SECSIDE : array [1..1000] of AB ;
00360     SIDE : boolean ;
00370     A : array [1..100, 1..50] of INOUY ;
00380     ABC : integer ;
00390     B : array [1..100] of SPAN ;
00400     TREE : array [1..50] of 1..50 ;
00410     X, Y : char ;
00420 procedure NOLAY ( I : integer ) ;{FINDS UNLAIN PATHS}
00430 begin
00440     WRITELN(OUT, SECSIDE[I].SIGNAL) ;
00450     WRITE(OUT, SECSIDE[I].POSITION)
00460 end
00470 procedure STRANERSTree ( STARTC, ENDC : integer ) ;
00480 {FINDS A PATH BET START AND END CELLS BY THIS METHOD}
00490 var

```

```

00500 I,P : integer ;
00510 procedure NEIGHBOURHOOD ; LATEST UPDATED CELLS\
00520 %FINDS EMPTY NEIGHBOR OF, LATEST UPDATED CELLS\
00530 var
00540   I,J,EG : integer ;
00550   procedure CHECK (A : integer) ;
00560   %CHECKS CELL AVAILABLE FOR UPDATING\
00570   begin
00580     if SIDE
00590     then
00600       begin
00610         if (CELLSTATUS[A].SELECT=0) or (CELLSTATUS[A].SELECT=V00FFPATH)
00620         then
00630           begin
00640             if CELLSTATUS[A].SELECT=0 then CELLSTATUS[A].SELECT := 1 ;
00650             LISTLPT2 := LISTLPT2+1 ;
00660             LISTLPT2 := A ;
00670             CELLSTATUS[A].CHAINCOORD := EG
00680           end
00690         else
00700           begin
00710             if CELLSTATUS[A].PIN=0
00720             then
00730               begin
00740                 LISTLPT2 := LISTLPT2+1 ;
00750                 LISTLPT2 := A ;
00760                 CELLSTATUS[A].CHAINCOORD := EG ;
00770                 CELLSTATUS[A].PIN := 2
00780               end
00790             else
00800               if (CELLSTATUS[A].PIN=1) and (CELLSTATUS[A].SELECT=V00FFPATH)
00810               then
00820                 begin
00830                   LISTLPT2 := LISTLPT2+1 ;
00840                   LISTLPT2 := A ;
00850                   CELLSTATUS[A].CHAINCOORD := EG
00860                 end
00870               end
00880             end
00890           end { check } ;
00900         begin
00910           for J := LISTLPT1+1 to P do
00920             begin
00930               EG := LISTL(J) ;
00940               I := EG-1 ;
00950               CHECK(I) ;
00960               I := EG+1 ;
00970               CHECK(I) ;
00980

```

```

00500 I,P : Integer ;
00510 procedure NEIGHBOURHOOD ;
00520 %FINDS EMPTY NEIGHBOR OF LATEST UPDATED CELL\N
00530 var
00540 I,J,EG : Integer ;
00550 procedure CHECK ( A : Integer ) ;
00560 %CHECKS CELL AVAILABLE FOR UPDATING\N
00570 begin
00580 if SIDE
00590 then
00600 begin
00610 if (CELLSTATUS[A].SELECT=0) or (CELLSTATUS[A].SELECT=NOFFPATH)
00620 then
00630 begin
00640 if CELLSTATUS[A].SELECT=0 then CELLSTATUS[A].SELECT := 1 ;
00650 LISTLPT2 := LISTLPT2+1 ;
00660 LISTLPT2 := A ;
00670 CELLSTATUS[A].CHAINCOORD := EG
00680 end
00690 end
00700 else
00710 begin
00720 if CELLSTATUS[A].PIN=0
00730 then
00740 begin
00750 LISTLPT2 := LISTLPT2+1 ;
00760 LISTLPT2 := A ;
00770 CELLSTATUS[A].CHAINCOORD := EG ;
00780 CELLSTATUS[A].PIN := 2
00790 end
00800 else
00810 if (CELLSTATUS[A].PIN=1) and (CELLSTATUS[A].SELECT=NOFFPATH)
00820 then
00830 begin
00840 LISTLPT2 := LISTLPT2+1 ;
00850 LISTLPT2 := A ;
00860 CELLSTATUS[A].CHAINCOORD := EG
00870 end
00880 end
00890 end { check } ;
00900 begin
00910 for J := LISTLPT1+1 to P do
00920 begin
00930 EG := LISTLPT1 ;
00940 I := EG-1 ;
00950 CHECK(I) ;
00960 I := EG+1 ;
00970 CHECK(I) ;
00980 I := EG+XCOORD ;

```

```

00990 CHECK(I);
01000 I := EG-XCOORD ;
01010 CHECK(I)
01020 end
01030 { neighbourhood } ; integer ) ;
01040 procedure BACKTRACK ( AC ; integer ) ;
01050 RETRACES THE PATH IF IT IS THERE\
01060 var
01070 IPOINT,XPOINT,X,B : integer ;
01080 begin
01090 X := AC ;
01100 XPOINT := 0 ;
01110 CELLSTATUS[AC].SELECT := NOOFFPATH ;
01120 CELLSTATUS[STARTC].SELECT := NOOFFPATH ;
01130 repeat
01140 R := X ;
01150 X := CELLSTATUS[B].CHAINCOORD ;
01160 if X <> 0
01170 then
01180 begin
01190 if not side then CELLSTATUS[X].PIN := 1 ;
01200 CELLSTATUS[X].SELECT := NOOFFPATH ;
01210 if (X=B-1) or (X=B+1) then IPOINT := 1
01220 else IPOINT := -1 ;
01230 if XPOINT <> IPOINT
01240 then
01250 begin
01260 XPOINT := IPOINT ;
01270 WRITELN(OUTPUT,B)
01280 end
01290 end
01300 until X=0 ;
01310 WRITELN(OUTPUT,STARTC) ;
01320 WRITELN(OUTPUT,0) ;
01330 end { backtrack } ;
01340 function PATH ( var AC : integer ) : boolean ;
01350 %VERIFIES WHETHER A PATH HAS BEEN FOUND\
01360 var
01370 X : integer ;
01380 PATH := false ;
01390 begin X := LISTLPTI+1 to LISTLPT2 do
01400 for if CELLSTATUS[LISTL(X)].SELECT=NOOFFPATH
01410 then
01420 begin
01430 PATH := true ;
01440 AC := LISTL(X)
01450 end
01460 { path } ;
01470 begin

```

```

01480 PRVIOUS := CELLSTATUS(ENDC1).SELECT ;
01490 CELLSTATUS(ENDC1).SELECT := NOFFPATH ;
01500 CELLSTATUS(STARTC1).CHAINCOORD := 0 ;
01510 if (ENDC=2) or (ENDC=3) then CELLSTATUS(ENDC1).SELECT := 1 ;
01520 if not SIDE 0 ;
01530 LISTLPT1 := 0 ;
01540 LISTLPT2 := 1 ;
01550 LISTL[1] := STARTC ;
01560 repeat
01570   P := LISTLPT2 ;
01580   NEIGHBOURHOOD ;
01590   LISTLPT1 := P
01600   until (PATH(AC)) or (LISTLPT1=LISTLPT2) ;
01610   if SIDE
01620     then for I := 1 to LISTLPT2 do CELLSTATUS(LISTL[I]).SELECT := 0
01630     else
01640       for I := 1 to LISTLPT2 do
01650         if CELLSTATUS(I).PIN=2 then CELLSTATUS(I).PIN := 0 ;
01660         CELLSTATUS(STARTC1).SELECT := 1 ;
01670         CELLSTATUS(STARTC1).SELECT := PRVIOUS ;
01680         CELLSTATUS(STARTC1).PIN := 1 ;
01690         CELLSTATUS(ENDC1).PIN := 1 ;
01700         if LISTLPT1=LISTLPT2
01710           then
01720             if SIDE
01730               then
01740                 begin
01750                   SECSID1:=SECP1+1 ;
01760                   SECSID1:=SECP1+1 ;
01770                   SECSID1:=SECP1+1 ;
01780                   SECSID1:=SECP1+1 ;
01790                   SECSID1:=SECP1+1 ;
01800                   SECSID1:=SECP1+1 ;
01810                   SECSID1:=SECP1+1 ;
01820                   SECSID1:=SECP1+1 ;
01830                   SECSID1:=SECP1+1 ;
01840                   end
01850                   else NOLAY(SECP1)
01860                   else BACKTRACK(AC)
01870                   end
01880                   { stannerstreet }
01890                   procedure CHAININGMETHOD ( STARTC, ENDC : integer ) ;
01900                   %FINDS A PATH BETWEEN START CELL TO END CELL BY THIS METHOD
01910                   var
01920                     P : integer ;
01930                     procedure NEIGHBOURHOOD ;
01940                     %FINDS NEXT CELL AVAILABLE FOR LIST UPDATED CELLS
01950                     var
01960                     I, J, EG : integer ;
01970                     procedure CHECK ( A : integer ) ;
01980                     %FINDS IF CELL CAN BE UPDATED IN LIST OR NOT

```

```

begin
  if SIDE
  then
    begin
      if (CELLSTATUS[A].SELECT=0)
      then
        begin
          CELLSTATUS[A].SELECT := 1 ;
          LISTLPT2 := LISTLPT2+1 ;
          LISTLILISTLPT2I := A ;
          CELLSTATUS[A].CHAINCOORD := EG
        end
      end
    else
      if CELLSTATUS[A].PIN=0
      then
        begin
          CELLSTATUS[A].PIN := 2 ;
          LISTLPT2 := LISTLPT2+1 ;
          LISTLILISTLPT2I := A ;
          CELLSTATUS[A].CHAINCOORD := EG
        end
      end { check } ;
    end
  begin J := LISTLPT1+1 to P do
  for
    begin
      EG := LISTLILJ ;
      I := EG-1 ;
      CHECK(I) ;
      I := EG+1 ;
      CHECK(I) ;
      I := EG+XCOORD ;
      CHECK(I) ;
      I := EG-XCOORD ;
      CHECK(I)
    end
  end { neighbourhood } ;
  procedure BACKTRACK ( AC : integer ) ;
  &RETRACES PATH IF IT IS THERE\
  var
    IPOINT,XPOINT,X,B : integer ;
  begin
    X := AC ;
    XPOINT := 0 ;
    repeat
      B := X ;
      X := CELLSTATUS[B].CHAINCOORD ;
      if X<>0
      then

```

```

01970
01980
01990
02000
02010
02020
02030
02040
02050
02060
02070
02080
02090
02100
02110
02120
02130
02140
02150
02160
02170
02180
02190
02200
02210
02220
02230
02240
02250
02260
02270
02280
02290
02300
02310
02320
02330
02340
02350
02360
02370
02380
02390
02400
02410
02420
02430
02440
02450

```

```

02950 CELLSTATUS[STARTC].SELECT := 1 ;
02960 CELLSTATUS[ENDC].SELECT := 1 ;
02970 if LISTLPT1=LISTLPT2
02980 then
02990   begin
03000     if SIDE
03010     then
03020       begin
03030         SECPT := SECPT+1 ;
03040         SECSIDE[SECPT].DATA := STARTC ;
03050         SECSIDE[SECPT].SIGNAL := ISIG ;
03060         SECSIDE[SECPT].POSITION := JPOS ;
03070         SECPT := SECPT+1 ;
03080         SECSIDE[SECPT].DATA := ENDC ;
03090         SECPT := SECPT+1 ;
03100         SECSIDE[SECPT].DATA := NOOFFPATH
03110       end
03120     else NOLAY(SECPT)
03130     end
03140   else BACKPACK(AC) ;
03150   end { chainingmethod } ;
03160 end { initialization } ;
03170 procedure INITIALISES SYSTEM ;
03180 var
03190   I : integer ;
03200 begin
03210   N := XCOORD*YCOORD ;
03220   for I := 1 to XCOORD do
03230     begin
03240       CELLSTATUS[I].SELECT := 1 ;
03250       CELLSTATUS[I].PIN := I ;
03260       CELLSTATUS[I+XCOORD*(YCOORD-1)].PIN := 1 ;
03270       CELLSTATUS[I+XCOORD*(YCOORD-1)].SELECT := 1
03280     end ;
03290     for I := 1 to YCOORD do
03300       begin
03310         CELLSTATUS[I*XCOORD].SELECT := 1 ;
03320         CELLSTATUS[I*XCOORD].PIN := I ;
03330         CELLSTATUS[(I-1)*XCOORD+1].PIN := 1 ;
03340         CELLSTATUS[(I-1)*XCOORD+1].SELECT := 1 ;
03350       end
03360     end { initialization } ;
03370 procedure POLINE ;
03380 $DRAWS POWER & GROUND LINES\
03390 var
03400   I : integer ;
03410 begin
03420   for I := 2 to YCOORD-1 do
03430     begin

```

```

03440 CELLSTATUS[(I-1)*XCOORD+21].SELECT := 2 ;
03450 CELLSTATUS[(I-1)*XCOORD+3].SELECT := 2 ;
03460 CELLSTATUS[I*XCOORD-21].SELECT := 3 ;
03470 CELLSTATUS[I*XCOORD-31].SELECT := 3
03480 end ;
03490 WRITELN(OUTPUT,XCOORD+TRUNC(XCOORD/2)-16) ;
03500 for I := 2 to TRUNC(XCOORD/2)-16 do
03510 begin
03520 CELLSTATUS[XCOORD+I].SELECT := 2 ;
03530 CELLSTATUS[2*XCOORD+I].SELECT := 2
03540 end ;
03550 for I := 2 to TRUNC(XCOORD/2)+16 do
03560 begin
03570 CELLSTATUS[2*XCOORD-I].SELECT := 3 ;
03580 CELLSTATUS[3*XCOORD-I].SELECT := 3
03590 end ;
03600 WRITELN(OUTPUT,XCOORD+2) ;
03610 WRITELN(OUTPUT,(YCOORD-2)*XCOORD+2) ;
03620 WRITELN(OUTPUT,0) ;
03630 WRITELN(OUTPUT,XCOORD+TRUNC(XCOORD/2)+16) ;
03640 WRITELN(OUTPUT,2*XCOORD-2) ;
03650 WRITELN(OUTPUT,(YCOORD-1)*XCOORD-2) ;
03660 WRITELN(OUTPUT,0) ;
03670 WRITELN(OUTPUT,'999996')
03680 end { polyline }
03690 procedure MIDORDERING ( AC : integer ) ;
03700 $ORDERS SIGNAL SET ACCORDING TO DISTANCE FROM CENTRE.
03710 var
03720 I,X,Y,J,XPOINT,YPOINT,MIDPOINT : integer ;
03730 M : boolean ;
03740 begin
03750 XPOINT := TRUNC(XCOORD/2) ;
03760 YPOINT := TRUNC(YCOORD/2) ;
03770 MIDPOINT := (YPOINT-1)*XCOORD+XPOINT ;
03780 I := 3 ;
03790 repeat
03800 J := 1 ;
03810 Bf11.Dist := 0 ;
03820 repeat
03830 XY := ABS(Af11,J).DATA-MIDPOINT) mod XCOORD+ARS
03840 (TRUNC(Af11,J).DATA/XCOORD)-YPOINT) ;
03850 if Bf11.Dist < XY then Bf11.Dist := XY ;
03860 J := J+1 ;
03870 until Af11.J.DATA=0 ;
03880 Bf11.ORDER := I ;
03890 I := I+1 ;
03900 until I=AC+1 ;
03910 repeat
03920 M := false ;

```



```

03930 for I := 3 to AC-1 do
03940   begin
03950     if B[I].DIST > B[I+1].DIST
03960     then
03970       begin
03980         Y := B[I].DIST;
03990         B[I].DIST := B[I+1].DIST;
04000         B[I+1].DIST := Y;
04010         Y := B[I].ORDER;
04020         B[I].ORDER := B[I+1].ORDER;
04030         B[I+1].ORDER := Y;
04040         M := true;
04050       end
04060     end
04070   until M = false
04080   end { midordering }
04090   procedure AREAORDERING ( AC : integer );
04100   { ORDERS SIGSET ACCORDING TO AREA SPANNED }
04110   var
04120     Y, X, I, XPOINT1, XPOINT2, YPOINT1, YPOINT2 : integer;
04130     M : boolean;
04140     begin
04150       I := 3;
04160       repeat
04170         J := 3;
04180         YPOINT1 := A[I,1].DATA;
04190         XPOINT1 := A[I,1].DATA;
04200         YPOINT2 := A[I,2].DATA;
04210         XPOINT2 := A[I,2].DATA;
04220         B[I].XSPAN := ABS(XPOINT1-XPOINT2) mod XCOORD;
04230         B[I].YSPAN := ABS(YPOINT1-YPOINT2) mod XCOORD;
04240         while A[I,J].DATA < 0 do
04250           begin
04260             if B[I].XSPAN < ABS(XPOINT1-A[I,J].DATA) mod XCOORD
04270             then
04280               if ABS(XPOINT1-A[I,J].DATA) mod XCOORD > ABS(XPOINT2-A[I,J].DATA)
04290               mod XCOORD
04300               then
04310                 begin
04320                   XPOINT2 := A[I,J].DATA;
04330                   B[I].XSPAN := ABS(XPOINT1-A[I,J].DATA) mod XCOORD;
04340                 end
04350               else
04360                 begin
04370                   XPOINT1 := A[I,J].DATA;
04380                   B[I].XSPAN := ABS(XPOINT2-A[I,J].DATA) mod XCOORD;
04390                 end
04400               else
04410                 if B[I].XSPAN < ABS(XPOINT2-A[I,J].DATA) mod XCOORD

```

```

044200 then
044300   begin
044400     XPOINT1 := A(I,J).DATA ;
044500     B(I).XSPAN := ABS(XPOINT2-A(I,J).DATA) mod XCOORD
044600   end ;
044700   X := ARS(TRUNC(XPOINT1/XCOORD))-TRUNC(A(I,J).DATA/XCOORD) ;
044800   Y := ARS(TRUNC(YPOINT2/XCOORD))-TRUNC(A(I,J).DATA/XCOORD) ;
044900   if B(I).YSPAN<X
045000   then
045100     if X>Y
045200     then
045300       begin
045400         YPOINT2 := A(I,J).DATA ;
045500         B(I).YSPAN := X
045600       end
045700     else
045800       begin
045900         XPOINT1 := A(I,J).DATA ;
046000         B(I).YSPAN := Y
046100       end
046200     else
046300       if Y>B(I).YSPAN
046400       then
046500         begin
046600           YPOINT1 := A(I,J).DATA ;
046700           B(I).YSPAN := Y
046800         end ;
046900         J := J+1
047000       end ;
047100       B(I).ORDER := I ;
047200       B(I).DIST := B(I).XSPAN*B(I).YSPAN ;
047300       I := I+1
047400       until I=AC+1 ;
047500       repeat
047600         M := false ;
047700         for I := 3 to AC-1 do
047800           begin
047900             if B(I).DIST>B(I+1).DIST
048000             then
048100               begin
048200                 Y := B(I).DIST ;
048300                 B(I+1).DIST := Y ;
048400                 B(I+1).ORDER := I ;
048500                 Y := B(I).ORDER ;
048600                 B(I+1).ORDER := B(I+1).ORDER ;
048700                 B(I+1).ORDER := Y ;
048800                 M := true
048900               end
049000             end
049100           end
049200         end
049300       end
049400     end
049500   end
049600 end

```

```

04910 until M=false
04920 end { areordering } ;
04930 procedure NEAREST ( var KJ : integer ) ;
04940 *FINDS PIN NEAREST TO ALL PINS CONSIDERED SO FAR
04950 type
04960 SPANNINGTree = record
04970     DIST, STARTC, ENDC : integer
04980 end ;
04990 var
05000 K, I, J, DIST : integer ; SPANNINGTree :
05010 R : array [1..50] of SPANNINGTree ;
05020 begin
05030     J := 2 ;
05040     K := 0 ;
05050     repeat
05060         I := 1 ;
05070         MIN := N ;
05080         if A[ABC, I].SELECT=0
05090         then
05100             begin
05110                 K := K+1 ;
05120                 repeat
05130                     DIST := ABS(A[ABC, TREE[I]].DATA-A[ABC, J].DATA) mod XCOORD+
05140                     ABS(TRUNC(A[ABC, TREE[I]].DATA/XCOORD)-
05150                     TRUNC(A[ABC, J].DATA/XCOORD)) ;
05160                     if MIN>DIST
05170                     then
05180                         begin
05190                             R[K].DIST := DIST ;
05200                             R[K].STARTC := TREE[I] ;
05210                             R[K].ENDC := J ;
05220                             MIN := DIST
05230                         end ;
05240                         I := I+1
05250                     until TREE[I]=0
05260                 end ;
05270                 J := J+1
05280             until A[ABC, J].DATA=0 ;
05290             J := 1 ;
05300             MIN := B[I].DIST ;
05310             if K<>1
05320             then
05330                 begin
05340                     for I := 2 to K do
05350                         begin
05360                             if MIN>B[I].DIST

```

```

05400 J := 1
05410 end
05420 end
05430 STARTC := B[J].STARTC ;
05440 ENDC := B[J].ENDC ;
05450 KJ := K nearest ;
05460 end
05470 begin %LAYOUT%
05480 I := RUNTIME ;
05490 REWRITE(OUT) ;
05500 READ(INPUT,XCOORD,YCOORD) ;
05510 READ(INPUT,I) ;
05520 repeat
05530 CELLSTATUS(I).SELECT := 1 ;
05540 CELLSTATUS(I).PIN := 1 ;
05550 READ(I)
05560 until I=99998 ;
05570 I := 0 ;
05580 SECT:=0 ;
05590 repeat
05600 I := I+1 ;
05610 J := 0 ;
05620 repeat
05630 J := J+1 ;
05640 READ(INPUT,A[I,J].DATA) ;
05650 A[I,J].SELECT := 0
05660 until (A[I,J].DATA=0) or (A[I,J].DATA=99999)
05670 until A[I,J].DATA=99999 ;
05680 INITIALIZE ;
05690 POLINE ;
05700 I := 1 ;
05710 NOFFPATH := 2 ;
05720 SIDE := true ;
05730 while A[I,I].DATA<>0 do
05740 begin
05750 STARTC := A[I,I].DATA ;
05760 ENDC := 2 ;
05770 ISIG := 1 ;
05780 JPOS := I ;
05790 STRANERSTree(STARTC,ENDC) ;
05800 I := I+1
05810 end ;
05820 I := 1 ;
05830 NOFFPATH := 3 ;
05840 while A[I,I].DATA<>0 do
05850 begin
05860 STARTC := A[I,I].DATA ;
05870

```

```

05890 ENDC := 3 ;
05900 ISIG := 2 ;
05910 JPOS := 1 ;
05920 STRANERSTree(STARTC,ENDC) ;
05930 I := I+1
05940 end ;
05950 NOOFFPATH := 5 ;
05960 READ(IN, X) ; MIDORDERING(NSIGNAL)
05970 if X= AREADORDERING(NSIGNAL) ;
05980 READ(IN, Y) ;
05990 if Y= B, then ISIG := NSIGNAL
06000 else ISIG := 3 ;
06010 READ(IN, X) ;
06020 repeat
06030 JPOS := 1 ;
06040 ABC := B(ISIG, ORDER) ;
06050 A(ABC, JPOS) := 1 ;
06060 TREE(JPOS) := 1 ;
06070 TREE(JPOS+1) := 6 ;
06080 JPOS := JPOS+1 ;
06090 repeat
06100 NEAREST(KJ) ; ENDC ;
06110 TREE(JPOS+1) := 0 ;
06120 A(ABC, ENDC) := SELECT := 1, DATA ;
06130 ENDPT := A(ABC, STARTC) . DATA ;
06140 START := A(ABC, ENDC) . DATA ;
06150 JPOS := JPOS+1 ;
06160 if X= C, then CHAININGMETHOD(START, ENDPT)
06170 else STRANERSTree(START, ENDPT)
06180 until KJ=1 ;
06190 if Y= B, then ISIG := ISIG-1
06200 else ISIG := ISIG+1 ;
06210 NOOFFPATH := NOOFFPATH+1
06220 until (ISIG=NSIGNAL+1) or (ISIG=2) ;
06230 SIDE := false ;
06240 WRITELN(OUTPUT, '99997') ;
06250 SECPT := 1 ;
06260 READ(IN, Y) ;
06270 if then
06280 begin
06290 while SECSIDE[SECPT].DATA<>0 do
06300 begin
06310 while
06320 begin
06330 NOLAY(SECPT) ;
06340 SECPT := SECPT+3
06350 end
06360 end
06370

```

```

06380
06390
06400
06410
06420
06430
06440
06450
06460
06470
06480
06490
06500
06510
06520
06530

else
begin
  while SECSIDE[SECP1].DATA<>0 do
  begin
    STARTC := SECSIDE[SECP1].DATA ;
    ENDC := SECSIDE[SECP1+1].DATA ;
    NOOFFPATH := SECSIDE[SECP1+2].DATA ;
    if X=C then CHAININGMETHOD(STARTC,ENDC)
    else STRANESTREE(STARTC,ENDC) ;
    SECP1 := SECP1+3
  end
end ;
T2 := RUNTIME ;
WRITE(TTY,T2,T1,T2-T1) ;
WRITELN(OUTPUT,'99999')
end .

```

	74
	62
0	4682
	106
	118
0	4738
99996	
	2043
	2055
	2115
0	2120
	2120
	2060
	2081
	2141
0	2146
	133
	613
	615
	675
0	680
	2145
	3525
0	3526
	3663
	3675
	3735
0	3740
	2145
	705
0	706
	373
0	372
	168
0	408
	3597
	3584
	2924
	2899

0 2839
2834
0 2920
2860
140
620
621
1341
1339
1399
1394
0 3584
3583
4243
4240
0 2901
3921
3919
3979
3974
0 1341
1342
1402
1420
0 920
919
1279
1278
1338
1333
1453
1481
641
647
1067
1066
0 2380
2379
2439
2419
2479
2474

0

2594
2593
3433
3435
3495
3500

0

402
403
583
588
2388
2386

0

3646
3645
4305
4299
4119
4100

0

2140
2200
2201
2681
2659
2719
2714

0

3854
3853
4033
4041
3981
3982
3742
3760

0
99997

3734
3674
3673
2233
2234

0

3640
3580
3581

0 1301
1300

3766
3826
3819
3639
3620

0

826

766

765

465

443

383

381

0

946

886

837

347

326

386

384

0

1060

1000

1001

941

942

882

883

523

502

442

440

380

378

0

1180

1240

1242

1302

1304

1244

1247

1067

1068

1008

1009
949
950
890
891
291
265
325
317
377
375

5

99999⁰

```

00100          1
00200          DIMENSION V(4),W(4)
00300          INTEGER A,B,XSPAN,YSPAN,XA,YA,XB,YB,I,J
00400          DATA V/0.0,0.75,0.0,1.0/
00500          DATA W/0.0,60.0,0.0,80.0/
00600          ACCEPT *,J
00700          CALL NITDEV(3)
00800          CALL WINDOW(W)
00900          CALL VPORT(V)
01000          CALL SELDEV(3)
01100          1 READ(20,*)XSPAN,YSPAN
01200          READ(20,*)A
01300          IF(A.EQ.99998)GO TO 2
01400          Y = FLOAT(A)/FLOAT(XSPAN)
01500          I=Y
01600          X= A - I*XSPAN
01700          Y = YSPAN -I
01800          CALL LINE (X,Y+0.2,0)
01900          CALL LINE (X,Y-0.2,1)
02000          GO TO 1
02100          2 READ(21,*)A
02200          IF(A.EQ.99999.OR.A.EQ.99997)GO TO 3
02300          IF (A.EQ.99996) GO TO 2
02400          4 READ(21,*)B
02500          IF (B.EQ.0)GO TO 2
02600          YA = FLOAT(A)/FLOAT(XSPAN)
02700          XA = A - YA*XSPAN
02800          YA = YSPAN - YA
02900          YB =FLOAT(B)/FLOAT(XSPAN)
03000          XB = B - YB*XSPAN
03100          YB = YSPAN-YB
03200          WRITE(24,*)XA,YA,XB,YB
03300          CALL LINI (XA,YA,0)
03400          CALL LINI(XB,YB,1)
03500          A=B
03600          GO TO 4
03700          3 CONTINUE
03800          ACCEPT*,J
03900          IF(A.EQ.99997) GO TO 2
04000          STOP
          END

```

1
SO PLACE.PAS

R 60

XSPAN = 60; YSPAN = 80;

R 950

W[2] := 60.0; V[2] := 0.75;

R 970

W[4] := 80.0; V[4] := 0.75;

EB

RENAME INPUT=PLA1.IN

R GPAS

PLACE.PAS

ASS TTY 3

EX PLACE,SYS:PASLNK,/SEA SYS:GPGS,FORLIB

2

```
DEL ROU1.OUT
DEL OUT1
DEL PLA1.OUT
  RENAME PLA1.IN=INPUT
RENAME INPUT=OUTPUT
R PASREL
ROUTE.PAS
EX ROUTE.REL
```

```
RENAME PLA1.OUT=INPUT
RENAME ROU1.OUT=OUTPUT
RENAME OUT1=OUT
```

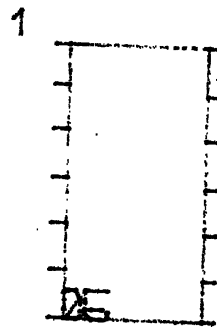
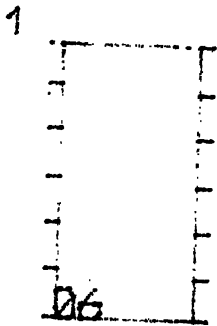
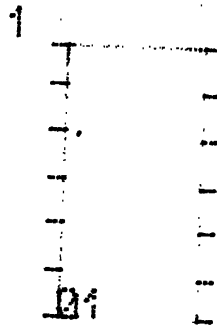
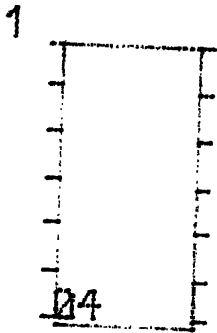
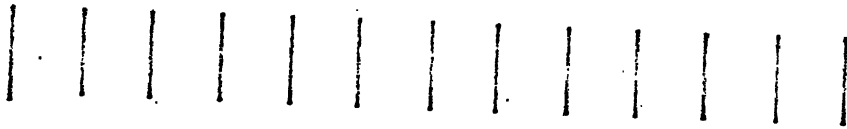
00100
00200
00300
00400
00500
00600
00700
00800
00900
01000

3
SU PLOT.FOR
R 300
* data V/0.0,0.75,0.0,1.0/
R 400
* data W/0.0,60.0,0.0,80.0/
EB
RENAME FOR20.DAT=PLA1.OUT
RENAME FOR21.DAT=ROU1.OUT
ASS TTY 3
EX PLOT.FOR,/SEA SYS:GPGS

00100 rename pla1.out=for20.dat
00200 rename rout.out=for21.dat
00300 REWAME PLA1.IN=INPUT
00400 DEL OUTPUT

4

EXIT

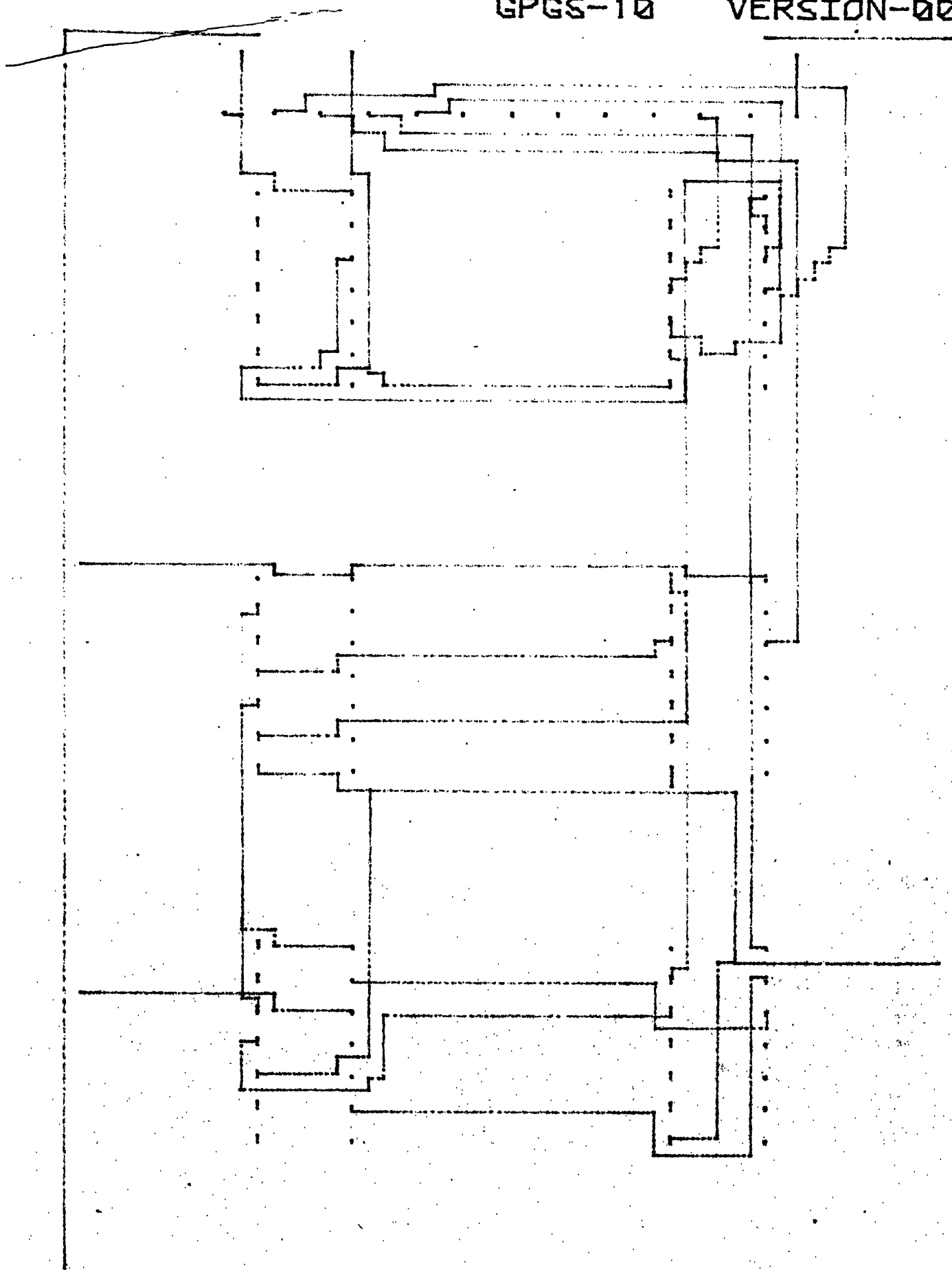


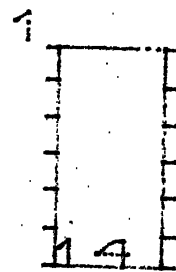
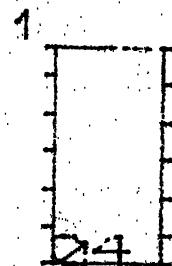
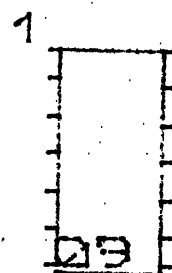
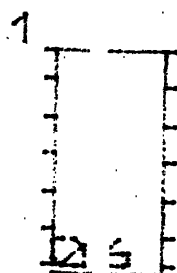
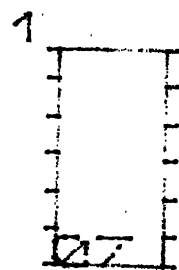
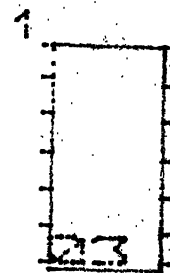
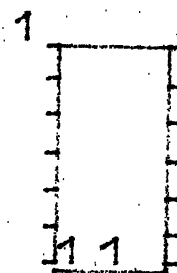
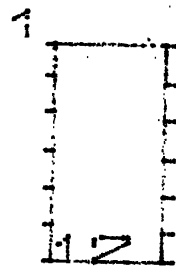
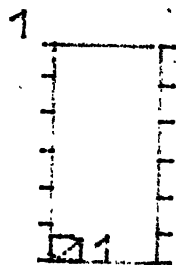
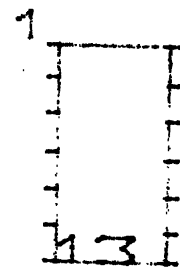
07

08

GPGS-10

VERSION-001





GPSS-10

VERSION-001

TE KIMURA
08-9-7

